PB-239 896

ROUTING OF SOLID WASTE COLLECTION VEHICLES

APPENDIX B: A HEURISTIC SOLUTION TO THE M-POSTMEN'S PROBLEM

ILLINOIS UNIVERSITY

PREPARED FOR
NATIONAL ENVIRONMENTAL RESEARCH CENTER

DECEMBER 1974

DISTRIBUTED BY:
NTIS
National Technical Information Service
U.S. DEPARTMENT OF COMMERCE

TECHNICAL REPORT DATA
(Please read Instructions on the reverse before completing)

1. REPORT NO.	2.	PB 239 899
EPA-670/2-74-036b		

4. TITLE AND SUBTITLE	5. REPORT DATE
ROUTING OF SOLID WASTE COLLECTION VEHICLES; Final Report. Appendix B: A Heuristic Solution to the M-Postmen's Problem.	December 1974-Issuing Date
	6. PERFORMING ORGANIZATION CODE
7. AUTHOR(S)	8. PERFORMING ORGANIZATION REPORT NO.
Jon C. Liebman and James W. Male	
9. PERFORMING ORGANIZATION NAME AND ADDRESS	10. PROGRAM ELEMENT NO.
University of Illinois Urbana, Illinois 61801	1DB314;ROAP 02AAE;TASK 04
	11. CONTRACT/GRANT NO.
	R801289
12. SPONSORING AGENCY NAME AND ADDRESS	13. TYPE OF REPORT AND PERIOD COVERED
National Environmental Research Center Office of Research and Development U.S. Environmental Protection Agency Cincinnati, Ohio 45268	Final
	14. SPONSORING AGENCY CODE

15. SUPPLEMENTARY NOTES

See EPA-670/2-74-035a, EPA-670/2-74-035b, and EPA-670/2-74-036a

16. ABSTRACT

This report describes research on computer-based and manual techniques for the routing of solid waste collection vehicles. Algorithms are given for the routing of single vehicles under three conditions: (1) all streets are two way, (2) all street are one way, and (3) some streets are two way and others are one way. In addition, a heuristic algorithm is provided for routing several vehicles in a larger area. Appendix A is a manual for use of the computer codes. Appendix B is a heuristic solution to the m-postmen's solution.

17. KEY WORDS AND DOCUMENT ANALYSIS

a. DESCRIPTORS	b. IDENTIFIERS/OPEN ENDED TERMS	c. COSATI Field/Group
*Collection, *Routing, Refuse, *Computer programming, Management, Computers, *Vehicles, Mathematical models, Algorithms, Heuristic methods	Solid waste, Solid waste management, *Solid waste collection, *Computing routing, Hauristic routing	13B

18. DISTRIBUTION STATEMENT	19. SECURITY CLASS *(This Report)*	21. NO. OF PAGES
Release to public	UNCLASSIFIED	130
	20. SECURITY CLASS *(This page)*	22. PRICE
	UNCLASSIFIED	5.75

EPA Form 2220-1 (9-73)

Reproduced by
NATIONAL TECHNICAL
INFORMATION SERVICE
US Department of Commerce
Springfield, VA. 22151

PRICES SUBJECT TO CHANGE

071060

EPA-670/2-74-036b
December 1974

ROUTING OF SOLID WASTE COLLECTION VEHICLES

Final Report

Appendix B: A Heuristic Solution to the M-Postmen's Problem

By

Jon C. Liebman, Principal Investigator,
and James W. Male
University of Illinois
Urbana, Illinois 61801

Program Element No. 1DB314

Project Officer

Donald A. Oberacker
Solid and Hazardous Waste Research Laboratory
National Environmental Research Center
Cincinnati, Ohio 45268

NATIONAL ENVIRONMENTAL RESEARCH CENTER
OFFICE OF RESEARCH AND DEVELOPMENT
U.S. ENVIRONMENTAL PROTECTION AGENCY
CINCINNATI, OHIO 45268

ia

REVIEW NOTICE

The National Environmental Research Center--Cincinnati has reviewed this report and approved its publication. Approval does not signify that the contents necessarily reflect the views and policies of the U.S. Environmental Protection Agency, nor does mention of trade names or commercial products constitute endorsement or recommendation for use.

FOREWORD

Man and his environment must be protected from the adverse effects of pesticides, radiation, noise and other forms of pollution, and the unwise management of solid waste. Efforts to protect the environment require a focus that recognizes the interplay between the components of our physical environment--air, water, and land. The National Environmental Research Centers provide this multidisciplinary focus through programs engaged in

- studies on the effects of environmental contaminants on man and the biosphere, and
- a search for ways to prevent contamination and to recycle valuable resources.

As part of these activities, this report describes research on computer based and manual techniques for the routing of solid waste collection vehicles.

A. W. Breidenbach, Ph.D.
Director
National Environmental
Research Center, Cincinnati

ABSTRACT

This paper presents the theory and procedure of an algorithm which routes a known number of solid waste collection vehicles on a city street network. The network is assumed to be planar and connected and consists of only undirected edges (two-way streets). Each edge has a cost and load associated with it.

The purpose of the procedure is to route the vehicles from a depot to a collection area (district), through the district and back to the depot. The load on each edge must be collected and the sum of the loads for one district must not exceed truck capacity. In order for the trucks to get from the depot to the districts, some edges may have to be traveled without being collected. The objective of the algorithm is to minimize the sum of the costs of these retraced edges.

The algorithm determines districts and tours simultaneously. For each tour it lists a sequence of nodes and specifies whether the edge leading to each node is traveled or serviced. Computational experience shows that the algorithm is feasible for a network with as many as 90 nodes.

PREFACE

The study of techniques for routing solid waste collection vehicles, of which this paper constitutes a portion of the final report, has been supported by Grant EP 801289 of the Solid Waste Research Division of the U. S. Environmental Protection Agency.

This appendix constitutes the doctoral dissertation of Dr. James W. Male, submitted to the Johns Hopkins University. Dr. Male is currently Assistant Professor in the Department of Environmental Engineering at Illinois Institute of Technology.

Jon C. Liebman
Principal Investigator

TABLE OF CONTENTS

		Page
ABSTRACT		iv
PREFACE		v
TABLE OF CONTENTS		vi
LIST OF TABLES		viii
LIST OF FIGURES		viii
CHAPTER		
1	INTRODUCTION.....	1
2	LITERATURE REVIEW.....	7
	2.1 The Chinese postman's problem.....	7
	2.2 Routing algorithms.....	11
	2.3 The m-postmen's problem.....	12
3	CHARACTERISTICS AND THEORETICAL BACKGROUND OF THE HEURISTIC.....	19
	3.1 Statement of the problem.....	19
	3.2 General outline of heuristic.....	20
	3.3 Theoretical basis.....	22
	3.3.1 Useful results from the Chinese postman's problem.....	22
	3.3.2 Cycles on a unicursal graph.....	24
	3.3.3 Network of cycle nodes.....	29
	3.3.4 Capacitated minimum spanning tree.....	32
	3.3.4.1 Branching procedure to find optimal arms.....	33
	3.3.4.2 Feasibility criterion for arms.....	42
	3.3.4.3 Building subtrees.....	43
	3.3.4.4 Balancing district loads.....	44
4	HEURISTIC ALGORITHM.....	46
	4.1 Detailed outline.....	46
	4.2 Lack of a guarantee of optimality.....	47
	4.3 Notes on the heuristic.....	51

		Page
5	RESULTS AND DISCUSSION.....	54
	5.1 Computational results.....	54
	5.2 Practical potential.....	57
	5.3 Implementation.....	59
	5.4 Possible modifications of the algorithm.....	63
	5.5 Extensions and other applications....	64
6	SUMMARY AND CONCLUSIONS.....	66
APPENDIX.....		71
REFERENCES.....		118

LIST OF TABLES

Table		Page
1	Computational Results.....	55

LIST OF ILLUSTRATIONS

Figure		Page
1	Example of Two Sets of Cycles on a Unicursal Graph.....	28
2	Example Showing the Creation of a Unicursal Graph and Designation of Cycles.....	30
3	Cycle Node Network and Decision Tree Showing 1-2-3 Set of Arms.....	34
4	Cycle Node Network and Decision Tree Showing 1-2-4 Set of Arms.....	36
5a	Cycle Node Network Showing 1-3-4 Set of Arms.....	38
5b	Extended Decision Tree with 3 Branches from the Decision Node Representing Arms 1-3-4.....	39
6	Entire Decision Tree for 6 Nodes taken 3 at a Time.....	41
7	Example of the Heuristic and Optimal Solution on a Small Network.....	49
8	Steps in Solving the Example Problem.....	116
8c	Final Tours of Example Problem.....	117

CHAPTER ONE

INTRODUCTION

The purpose of this study is to develop an efficient routing scheme for solid waste collection vehicles on a city street network. The need for such a scheme arises in any major city which has municipal solid waste management. Collection, treatment, and final disposal of solid waste use a substantial portion of a city's operating budget. Black (4) reports that 3 billion dollars are spent annually on solid waste management in the United States. In Baltimore, Maryland, with a population of approximately one million people, 4 million dollars are spent each year for collection and disposal. It is estimated that 75 to 85 percent of the total cost of managing solid waste is attributable to collection. Thus, an efficient use of collection vehicles would have a significant impact in reducing the cost of solid waste collection.

Current methods used to determine collection areas are generally trial and error by nature. The city street network may be subdivided into logical collection areas and then the collection areas are arbitrarily subdivided into districts capable of being collected by one truck. These districts, however, are often non-contiguous and inefficiently arranged on the network. As the city grows and

changes take place the districts are modified and increased with no clear objective in mind. Reallocating districts each time the network changes becomes a tedious task, proving not to be worthwhile when done by hand. Implementation of an efficient computer algorithm, however, would allow rapid analysis of the new network configuration whenever changes necessitate a modification of the districts.

The overall planning of a solid waste collection system involves decisions on a wide variety of issues. Long range planning decisions involve such questions as: the number and location of final disposal sites, the possible use and location of transfer stations, and the number and size of trucks. These questions involve planning projections and capital outlay. This study is limited to the short range decisions related to the routing of trucks from one depot through a collection area and back to the depot. The depot, or dispatching center, and disposal site (or transfer station) are fixed and assumed to be the same. The number of trucks is known and all capacities are assumed to be equal. Such questions as collection frequency and manpower scheduling are not considered. Changes in both of these practices may be desirable in consideration of the resulting routes.

Currently, solution procedures usually subdivide the street network and route one truck on each of the

subdivisions. It would be advantageous to simultaneously subdivide and route the trucks on the entire network.

In Operations Research literature a similar problem is known as the m-postmen's problem. It has application in several areas and an efficient solution technique would be very beneficial.

A characteristic of the collection problem is that demand for solid waste collection is located along street segments, rather than at intersections. Collection vehicles must travel all streets and collect as they travel from one intersection to another. The problem is to route a fleet of collection vehicles such that each street in the network has a collection service, capacity of the vehicles is not exceeded, and the total route cost is minimized.

Certain assumptions must be made concerning the characteristics of the street network. The procedure described in this paper is applicable only to a network with undirected streets; one-way streets are not allowed. The network must also be planar. If one street crosses another they must be joined; bridges cannot be considered. The network must be connected and the depot (disposal site or transfer station) is considered to be located at one of the street intersections.

The objective of the procedure is to minimize the

sum of the distances traveled by all of the trucks traveling from the depot through assigned collection areas, and back to the depot. Since the distance traveled during actual collection is a constant (the sum of the lengths of all the streets in the network), it is necessary to minimize only the distance traveled by the trucks when they are not collecting. These traveled streets are at one time serviced (collected) by another truck, or by the same one, and therefore are referred to as retraced streets. The objective is to minimize the total distance of retraced streets.

The distance or cost of the street segments can be any measure deemed appropriate by the planner. The actual length of the street could be used, or the time necessary to travel the street segment. Since the objective is generally to minimize the cost of solid waste collection, the most appropriate cost for each street segment would likely be a combination of the cost of operating the truck and the cost of maintaining a crew of men on the truck. Assume the crew is paid A dollars per hour and truck operating costs are B dollars per mile. The cost associated with a street segment would then be A times the time in hours necessary to traverse the segment plus B times the length of the segment, in miles. Each street segment has not only a cost but a load associated with it. The load

represents the amount of solid waste to be collected on that street segment, and may be measured in weight, or volume. The procedure described in chapter 4 is independent of the type of measure used for load and for cost.

There are two constraints imposed upon the solution. The first, mentioned earlier, is a real constraint. The solid waste collection vehicle has a set capacity; it must return to the depot when its capacity is met. Each round trip made by a truck will be called a tour. When one truck has completed two round trips it has serviced two tours. The tours which will be developed from the algorithm are independent and are not specified for certain trucks. The streets serviced in one tour comprise a district.

The second constraint requires that each district be contiguous. In other words, the serviced streets in a tour are not divided into two or more separate groups. In network theory terminology, the serviced edges in one district form a connected graph. The algorithm will allow a slight violation of this constraint in very rare circumstances, which will be discussed when the algorithm itself is developed.

The desirability of the contiguous district constraint could be challenged. Eliminating the constraint might result in a better solution, however, real world conditions must be considered. Most managers have found

contiguous, compact districts more appealing and less confusing to vehicle crews.

During construction of the districts the algorithm considers two other aspects. The first is the desirability of balanced districts. Districts which are balanced have approximately equal loads of solid waste. The second aspect is the desirability of compact districts. The term compact refers to the relative geographic location of the serviced edges in the district. Diverse, odd shaped districts are discouraged for the same reasons non-contiguous districts are discouraged. The algorithm is inclined toward balanced and compact districts during construction of the routes. However, neither consideration is used as feasibility or optimality criteria.

Throughout this paper reference will be made to terminology generally used in network theory. A network, or graph, is comprised of nodes and edges. Each edge is incident with exactly two nodes. Edges correspond to street segments and nodes represent intersections of streets. The depot, as described earlier, is a node of the network where all tours begin and end. A path is a sequence of edges and is usually denoted as being between two nodes which identify each end of the sequence. The shortest path between two nodes is the path which minimizes the sum of the edge costs in the path. If the two end nodes of a path are the same then the path is a cycle. A tour is therefore a cycle since the included edges form a path which begins and ends at the depot.

CHAPTER TWO

LITERATURE REVIEW

There has been a limited number of attempts to develop both heuristic and optimal solutions to the m-postmen's problem as it applies to solid waste management. These procedures take a variety of approaches and will be discussed in section 2.3. Crucial in understanding some of the solution approaches is an understanding of the Chinese postman's problem and related solution procedures. The Chinese postman's problem is identical to the m-postmen's problem except that one tour is found for the entire network. The Chinese postman's problem is useful when routing one truck over a predetermined collection area when the load on the area is less than the capacity of the truck. Solution procedures for the Chinese postman's problem are discussed in section 2.1.

A solution to the Chinese postman's problem generally determines only the necessary retraced edges and does not actually generate the sequence of edges comprising a tour. Section 2.2 describes two tour generating algorithms.

2.1. The Chinese postman's problem

The original work on the Chinese postman's problem was done by Euler (16) in 1736, when he was asked to route a parade over several bridges in the town of Königsberg.

The parade was to start and end at the same point and cross each bridge once and only once. Euler visualized a network where the city's islands and river banks became nodes and the bridges became undirected edges adjoining the appropriate nodes. He concluded that, since at least one node was incident with an odd number of edges, it was impossible to route the parade. His reasoning was this: a cycle, or tour must both enter and leave every node, requiring a pair of edges. An odd number of incident edges on any node results in a path for entering the node with no corresponding path for leaving. The theorem which resulted gives the evenness of incidence on every node as the necessary and sufficient condition for the existence of a cycle which traverses every edge once and only once. Such a cycle is called an euler tour, and a graph which has an euler tour is called unicursal. A node with an odd (even) number of incident edges will be referred to as an odd (even) node.

Kwan (22), in 1962, investigated a related optimization problem with an application to the walking tour of a mailman. His work appeared in the Chinese Journal of Mathematics and the name "Chinese postman's problem" has since evolved.

Kwan was concerned with the minimum length retracing of edges in order to construct a tour which traverses each edge at least once, in a network which is not unicursal.

Such a tour, which retraces some edges, is called a postman's tour to distinguish it from an euler tour which retraces no edges. It should be noted that a postman's tour in a particular graph is identical to an euler tour in an expanded graph where each of the retraced edges is duplicated. Kwan made three observations: (1) there is an even number of odd nodes, (2) pairwise matching of the odd nodes to create duplicate edges (retracings), will create a network with a feasible euler tour (a unicursal graph), and (3) the optimal postman's tour is the euler tour in the expanded graph with the minimum retraced length.

Kwan proposed an algorithm which determines the optimal solution. However, a total enumeration of all cycles in the network is necessary to guarantee optimality, and this is impractical in large networks.

Edmonds (9), in 1965, solved the problem of minimizing the number of retraced edges and also the total retraced distance. His algorithm is very efficient, making use of the primal-dual characteristics of a linear programming formulation.

When streets in the network are one-way, the result is a directed or a mixed graph. A directed graph has only one-way streets; a mixed graph has both one-way and two-way streets. The necessary and sufficient condition for unicursality in a directed graph is that the number of edges

directed toward each node must equal the number directed away from it.

The determination of a postman's tour in a directed network is simpler than in an undirected one. Each node with unequal incoming and outgoing edges is labelled with a number equal to the number of edges directed away from the node minus the number of edges directed toward the node. A bipartite network of positively and negatively labeled nodes is then formed. The edges of the new network represent the shortest paths from all negative nodes to all positive nodes. The minimum length combination of shortest paths is found, such that the unbalanced nodes become balanced when the arcs in the shortest paths are added to the original network.

The network of positive and negative nodes is more easily visualized as a transportation problem. The negative nodes become demand nodes with the demand on each node equal to the absolute value of the imbalance. The positive nodes become supply nodes with the supply equal to their imbalance. The Hitchcock transportation algorithm can then be used to solve for the minimum cost edges which represent retraced paths in the original network. The problem may also be solved, without first finding shortest paths, by applying the out-of-kilter algorithm (OKA) to the original graph.

Solving the Chinese postman's problem on a graph

with both directed and undirected edges proves more difficult than either of the directed or undirected cases. Wathne (33) has formulated a linear program to construct a minimal unicursal graph on a mixed network. The basis of the formulation is the requirement that the edges incident to each node must be balanced. The balance is different than those for the previous two cases because both directed and undirected edges must be considered. A unicursal graph on a mixed network must have an equal number of entries and exits on each node and each subset of nodes. If there is an imbalance of directed edges, the difference must be compensated for by undirected edges. To solve the linear program, Wathne uses a column generation procedure. The out-of-kilter algorithm is used to determine which vector enters the basis of the simplex tableau. The algorithm finds the optimal unicursal graph.

2.2. Routing algorithms

Edmonds and Johnson (15) have developed a scheme for generating a tour on undirected, directed, and mixed unicursal networks. The algorithm is based on the existence of a spanning arborescence on the network. A spanning arborescence is a spanning tree of directed edges with all edges directed toward a root node. If a spanning arborescence exists and is specified on the network, then a tour is

traced by selecting first undirected edges and directed edges not in the arborescence, and finally directed edges in the arborescence. If a spanning arborescence does not exist on the original network, one is constructed following a sequence of rules and then the above procedure is followed to determine a tour.

A second procedure is described for finding a tour on an undirected, unicursal graph. The scheme simply finds random cycles and adds them together at points where cycles share a common node. These optimal routing algorithms do not consider various additional requirements which often occur, such as left-turn restrictions, sequencing restrictions, etc. Two heuristic approaches which permit consideration of such constraints will be described in section 2.3.

2.3. The m-postmen's problem

Two attempts have been made at routing solid waste collection vehicles by solving the m-salesman traveling salesman problem. The problem is similar to the m-postmen's problem, except that each node must be visited by one of m trucks with no restriction on tracings of edges. The objective is to minimize total distance traveled. Marks (26) and Beltrami and Bodin (2) each route vehicles in this manner.

Marks considers each node to represent a small

collection area with edge lengths being the distance between centroids of the areas. The nodes are assumed to generate equal amounts of waste and each truck must visit an equal number of nodes. A complete network is set up and an additional node, representing the depot, is added to the network. Edges are constructed from this dummy node to all other nodes and vice versa. Each node is restricted to being serviced by only one truck and the depot is restricted to m visits, representing the number of trucks. The problem is then solved, using the out-of-kilter algorithm, ignoring the restriction that each truck must visit an equal number of nodes, a branch and bound procedure restricts certain variables and the out-of-kilter algorithm is used again. The procedure is repeated until a feasible solution is obtained.

Beltrami and Bodin used a modification of the Clark and Wright (7) algorithm to solve the m-salesman's problem on an undirected network. The Clark and Wright method starts with tours from the depot to each node and back again. It then reduces the number of tours to the appropriate number by combining pairs of nodes such that only one return trip is made to the depot for each pair. The potential savings for each configuration is calculated and stored in a list with savings from other combinations. Feasible pairings are then made, starting with the one with

the greatest savings. Beltrami and Bodin approximated the optimal solution to the traveling salesman's problem to improve the order of choosing the sequence of nodes to be combined.

Two procedures use routing criteria to route trucks and consequently determine districts. They heuristicly guide trucks over the network until capacity is met, return to the depot and then repeat the process until the network is completely serviced.

Shuster and Schur (30) attempt to minimize not only retraced distance but also the number of left-hand and u-turns. Two general operating rules are proposed. If streets are collected one side at a time, then it is preferable to make right hand turns, when possible, before going straight. Similarly, continuing in a straight line has preference over left hand turns. When both sides of the street are collected at the same time, the order of preference is straight, right, and then left hand turns. These preferences are followed subject to five guidelines: (1) it is desirable for the resulting districts to be compact, (2) work loads on the districts should be equalized, (3) heavily traveled streets should be avoided during rush hours, (4) the starting point of each tour should be close to the depot, and (5) dead end streets should be collected to the right wherever possible.

The operating rules are used in conjunction with a look-ahead system which uses the guidelines to override the preferred order of streets. The rules are followed until truck capacity is met and then a new tour is started. Shuster and Schur have not implemented their procedure for computer use. The scheme is meant to be done by hand. They point out that experience greatly enhances the obtained results.

Bodner, Cassell, and Andros (5) simulate a tour based on a random walk procedure. At each intersection the algorithm randomly chooses a leaving edge. If all incident edges at the current node have been serviced, then the tour proceeds to the nearest unserviced edge and continues to collect. The scheme continues until either capacity is met or an eight-hour work period has been completed. At this point the truck returns to the depot and starts a new tour, if the network has not been completely collected. The simulation is run many times and the solution with the least cost of retraced edges is kept as the best solution.

Marks and Liebman (27) proposed a modification to the above algorithm. Rather than proceed to the nearest unserviced edge, when the incident edges have been serviced at the current node, the tour should proceed to the nearest node which has an odd number of incident, uncollected edges. This change takes advantage of the properties described by Kwan.

A method which determines temporary districts, and then modifies the district boundaries to create tours, is presented by Stricker (31). He first determines a unicursal graph on the entire network, using one of the Chinese postman's algorithms. Arbitrary district boundaries are chosen such that the district loads do not violate capacity. Edges are then exchanged from one district to another until an euler tour exists within each district. The feasible districts and tours are found by trial and error. Stricker's algorithm does not, however, consider the travel distance between the resulting districts and the depot.

A major concern of many city managers is balancing work loads of collection crews. Brown (6) has an algorithm whose objective is to balance the times collection crews spend during one day. The algorithm starts by subdividing the city into areas called blocks. Brown used census blocks as the subdivided areas. The size of the chosen blocks is variable. Smaller blocks allow greater flexibility but increase the difficulty in obtaining a solution. A list of trial districts is then determined where each district is made up of one or more blocks. Each trial district is formed such that (1) the total load is greater than a set amount and less than the truck capacity, and (2) the blocks comprising the trial district are compact and contiguous. The trial districts in the list are not mutually exclusive

and might have overlapping blocks. From this list a series of plans is developed such that in each plan, each block is included in exactly one district. An integer program is then formulated to minimize the deviation of crew times from the average time required to service the network. Each truck is assigned one or more districts according to the optimal integer program solution.

Two solution procedures convert the m-salesmen's problem into a larger single salesman's problem. Hong (20) creates m-1 artificial depots and joins the m m-salesmen's tours into one single salesman's tour. He then modifies an existing integer programming solution procedure to solve the single salesman's problem. Orloff (28) has developed a general routing algorithm, which includes both the m-postmen's and m-salesmen's problem. Certain nodes and edges in the network may need to be serviced. If no nodes have any demand then the problem is identical to the m-postmen's problem. The procedure requires that the m origins for each tour be specified. The algorithm first transforms the problem to one of a single general routing problem (Chinese postman's problem when no nodes need to be serviced), by adding edges from an imaginary depot to the m origin nodes and requiring these edges to be serviced. A unicursal graph is then constructed by solving the general routing problem (Chinese postman's problem). Next, euler tours are found.

A branch and bound procedure is used if any of the m subtours comprising the euler tour exceed truck capacity. Once a feasible solution is found the algorithm terminates.

An optimal solution to the m-postmen's problem has been claimed by Owen (29). The technique is considered proprietary and, therefore, proof of optimality is not available.

CHAPTER THREE

CHARACTERISTICS AND THEORETICAL BACKGROUND OF THE HEURISTIC

3.1. Statement of the problem

The remainder of this paper will address itself to one specific problem; that of routing a known number of vehicles to service all the streets of a connected, planar, undirected network. One node is specified as the depot at which all tours must begin and end, and each edge has a cost and load associated with it. Each of the m trucks collects over specified edges and the sum of the loads on these edges must not exceed a specified capacity. As stated earlier the edges collected by one truck comprise a district. It is desirable that each district be contiguous so that no crew will service disjoint sections of the network.

The objective of the problem is to minimize the total distance driven by the trucks. Since all the edges must be driven at least once during collection, only repeated use need be considered in the minimization. To differentiate between driving an edge while servicing and simply driving the edge, the following terminology will be used. An edge is serviced, or collected, by a tour if the load on that edge is collected and therefore, added to the total load of that tour. An edge is traversed if it is used as a means of getting from one node to another. Its load is

collected by another truck, or the same truck, at another time. Because each edge is serviced only once, it may then be necessary for some edges to be traversed once or more.

Results of the heuristic will be a sequence of edges for each tour, with collected edges being differentiated from traversed edges.

3.2. General outline of heuristic

The general steps of the heuristic procedure are outlined in this section. Following the outline will be a theoretical discussion of sections of the algorithm.

To begin the algorithm the Chinese postman's problem is solved on the original street network (G_1). The resulting unicursal graph (G_2) will have some added edges representing retraced streets. These edges are assigned costs equal to those of the duplicated edges, and will require service even though their actual loads are zero.

A set of cycles is next chosen on G_2, following a certain set of rules, such that each edge is in one and only one cycle. The cycles correspond to city blocks, although not all city blocks are represented. If the represented city blocks were shaded the network would resemble a checkboard pattern. Two cycles are adjacent if they share a common node. In other words, the same node appears in both cycles.

A new network (G_3) is then created whose nodes

(called cycle nodes) correspond to the cycles on G_2. Each cycle node has a load equal to the sum of the loads on the edges in their corresponding cycle. For every pair of adjacent cycles on G_2 there corresponds an edge with zero cost on G_3. All possible pairs of adjacent cycles are represented by zero-cost edges on G_3. No edges exist between cycle nodes which represent non-adjacent cycles. An additional artificial node, representing the depot, is created and edges from it to all cycle nodes are added. The cost on these edges in G_3 represent the shortest paths on G_2 from the depot to the closest node in each cycle.

On G_3 a minimum cost capacitated spanning tree is found. The capacitated spanning tree has two requirements over and above those of a regular spanning tree. First, m edges leading from the depot must be in the tree. These edges, called arms, represent the traversed distance between the depot and each district. Representing the districts are subtrees of cycle nodes attached to each of the m arms. The second requirement is that the sum of the cycle node loads, in each subtree, be less than capacity. The capacitated spanning tree represents a feasible but not necessarily optimal solution to the original problem. The optimal solution, or minimum cost capacitated spanning tree, is that capacitated spanning tree with the least cost combination of m arms.

Using Edmonds and Johnson's algorithm for building tours on an undirected network, the minimum cost capacitated spanning tree is decoded into m tours. If the edge between two cycle nodes, on G_3, is in the tree then the corresponding two cycles, on G_2, are combined. The procedure is repeated until all the cycles are included.

3.3. Theoretical basis

3.3.1. Useful results from the Chinese postman's problem

The unicursal graph which solves the Chinese postman's problem (CPP) is the minimum cost solution for one truck to service a network. This solution also represents the lower bound for the m postmen's problem (MPP) on the same network. No feasible set of tours can be found which has a lower cost. This can be shown by contradiction. Assume the minimum cost solution to the CPP for a network has cost A. Also assume that the minimum cost solution to the MPP on the same network has cost B. If B is less than A then combining the m tours of the MPP solution into one tour would represent a cheaper solution to the CPP. This contradicts the original assumption and shows that no MPP solution can be less than a CPP solution on the same network.

An MPP solution might be identical to a CPP solution if it has certain characteristics. The solution must contain m subtours, each including the depot, and each with a load

less than capacity. For this to be true the depot must be incident to at least 2m edges. For a network of any reasonable size and m greater than three or four, the conditions are rarely met. Therefore, additional edges must be traveled, to meet the constraints, increasing the minimum cost.

The Edmonds and Johnson algorithm for finding a tour on an undirected unicursal graph provides a theoretical base for part of the m postmen's heuristic. To generate a tour, they randomly create cycles, using previously untraveled edges as they are needed and until the entire network is covered. Ingrained in the construction of the cycles is the fact that each edge can be in one and only one cycle. Rather than adding arbitrary cycles, predetermined cycles can be combined with no loss of generality. As with the arbitrary cycles, predetermined cycles must also include each edge once and only once, and must be chosen with this in mind. The algorithm proceeds exactly as before except that the cycles need not be created at each iteration. Instead, the tour is expanded by adding an adjacent predetermined cycle.

The m postmen's problem algorithm predetermines the cycles and then heuristically combines them in order to solve the overall problem.

3.3.2. Cycles on a unicursal graph

The MPP requires that all edges in the original graph be serviced. For a sizeable network the number of edges is quite large and the number of possible feasible combinations of edges into districts is even larger. To reduce the number of possible combinations and therefore computation time, edges are included in cycles and then the cycles are combined to form feasible districts. Cycles are chosen as the groups of edges because they most realistically represent the network. Two factors are most obvious. First, a group of edges should be contiguous in order to guarantee a contiguous district after the groups are combined. Second, the edges in each district in the final solution will form cycles, because each district will be a unicursal graph.

Since all edges require collection, all edges must be included in the set of cycles. The cycles will then require service and represent all edges from the original network. Each edge is also to be serviced only once. Thus the set of cycles should contain each edge only once. Each edge on the original network, therefore, must be in one and only one cycle.

In order to find a set of cycles which completely covers the network and which satisfies the previous requirements, every node must be even. This is obvious since pairs

of edges at each node are needed for each cycle. Odd nodes are, therefore, paired to enable the formation of cycles. The pairing is done by simply solving the CPP on the original network, G_1. The resulting network, G_2, will contain duplicate edges which require service even though their load is zero. This is necessary, although not necessarily optimal in the long run, in order that each edge be in only one cycle. Since the CPP solution finds the least cost retracing, the resulting solution represents the minimum cost network upon which the cycles can be built.

It is interesting to note that, if the cycles were constructed on G_1 (with odd notes) without first solving the CPP, it would be necessary to use some edges more than once. If the cost of these retraced edges was minimized then the retraced edges would be identical to the duplicated edges which were added to create G_2. Duplicating these edges, rather than retracing them, not only helps identify the cycles in the network but allows the edge with a non-zero load to be identified with a certain cycle.

There is no unique set of cycles which satisfies the requirements that each edge in G_2 be in one and only one cycle node. However, the set of cycles has bearing on the outcome of the algorithm and therefore is chosen with truck capacity, district contiguity, and district compactness in mind.

It is advantageous that no cycle enclose any part of another cycle. Allowing such a configuration would not violate the contiguous district constraint. However, the resulting districts would most likely be less compact.

To satisfy capacity requirements, the sum of the loads on the cycles in one district must not exceed truck capacity. The smaller the load on each cycle, the more varied the possible combinations to create a set of feasible districts. It is, therefore, advantageous to find a large number of cycles each with a small load.

The set of cycles is then chosen such that (1) each edge on G_2 is in one and only one cycle, (2) no cycle on G_2 encloses any portion of another cycle, and (3) the number of cycles is maximized.

One set of cycles, on G_2, that satisfies these criteria is a set of simple polygons. Three such cycles are shown on a small network in Figure 1a. On a city street network each cycle would correspond to the streets surrounding one block. However, not all blocks are represented as cycles. Coloring the blocks represented as cycles on G_2 will result in a checkerboard pattern. Colored blocks are adjacent at nodes only, since each edge is in only one cycle. Each node will be incident to exactly N/2 cycles, where N is the number of edges incident to the node in G_2.

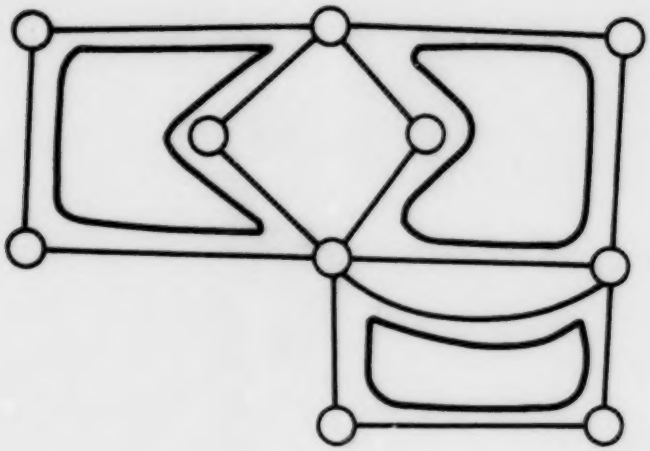
The checkerboard pattern of cycles always exists

since the network is both planar and unicursal. A unicursal network was defined as one which has an euler tour. The euler tour can be subdivided into cycles. If the cycles are subdivided such that no cycle encloses any part of another, then, since the network is planar, when no more cycles can be subdivided the checkerboard pattern has been attained.

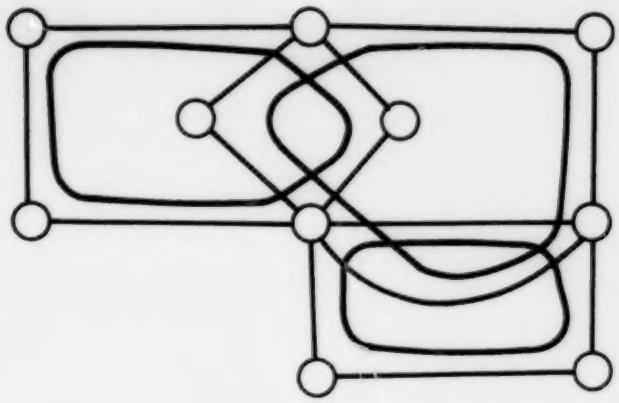
Another set of cycles which would satisfy the problem requirements is shown in Figure 1b. However, for simplicity and more compact districts the algorithm uses only the cycles as they are represented in Figure 1a. In other words, no path which is part of one cycle will ever be in the interior portion of another cycle.

There are two sets of cycles which represent the checkerboard pattern. One will be discarded because one cycle in the set includes all of the periferal edges of the network. If this set were used one truck would have to service the edges which outline the entire network. For a street network of reasonable size, this would most likely be infeasible because of the capacity constraint.

To construct the checkerboard cycles it is necessary to input a sequence of edges for each node. Each sequence simply lists, in clockwise order, the edges incident upon each node. The cycles are constructed by (1) choosing an unused edge and adding it to the cycle, (2) from a node incident to the previous edge choosing the next edge in the



(a) Cycles Determined By The Algorithm



(b) Feasible Cycles Not Used By The Algorithm

FIGURE 1. EXAMPLE OF TWO SETS OF CYCLES ON A UNICURSAL GRAPH

sequence, for that node, and adding it to the cycle, and (3) repeating the procedure until the first edge is encountered. Proper choice of the first edge will ensure that none of the encountered edges have previously been included in a cycle. The three steps are repeated until all edges are included in a cycle.

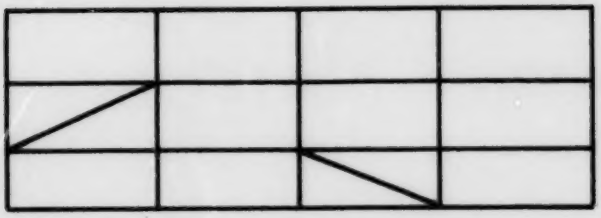
Since some of the edges on G_2 are duplicate edges, it is possible for an edge and its duplicate to be represented as a cycle node on G_3.

Figure 2a shows an original street network G_1. Figure 2b shows G_2 (G_1 with the matched edges added) and Figure 2c shows the checkerboard pattern of cycles on G_2.

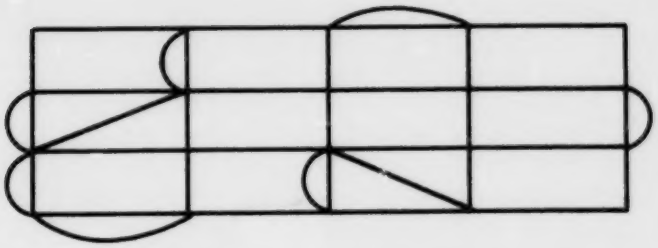
3.3.3. Network of cycle nodes

Once the original network has been matched and had cycles designated, a new network, G_3 is created. G_3 has nodes, called cycle nodes, for all designated cycles on G_2. Each cycle node is given a load equal to the sum of the loads of the edges in the cycle. The included nodes and therefore edges of G_2 are stored for each cycle so that the meaning of the cycle nodes in G_3 is not lost relative to G_1. After G_3 is manipulated this information is used to decode the results into tours on G_1. It is not necessary to store the distances of the edges in the cycle node because the sum of the distances is constant.

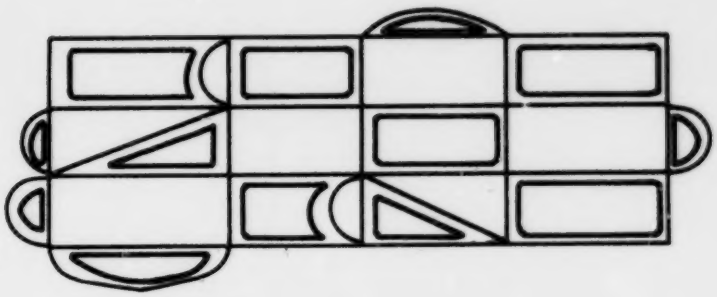
The edges on G_3 will represent any point on G_2 where two cycles are adjacent; in other words, where they share a



(a) Original Network



(b) Unicursal Graph, Formed By Adding Duplicate Edges



(c) Checkerboard Pattern Of Cycles

FIGURE 2. EXAMPLE SHOWING THE CREATION OF A UNICURSAL GRAPH AND DESIGNATION OF CYCLES

common node. The edges between cycle nodes on G_3 will have zero cost, since the distance between the two cycles on G_1 is zero.

One more node is created to represent the depot. This artificial node has an edge to each of the cycle nodes with a cost representing the shortest path, on G_2, from the depot to the closest node in the corresponding cycle. At least one of these edges will have zero cost because the depot is, in fact, the closest node of at least one cycle. If the depot has N edges incident with it, on G_2, then G_3 will have N/2 zero cost edges incident to the depot.

G_3 now completely represents G_2. Each cycle node, in G_3, has associated with it a load and a list of nodes contained in it. The edges connect all adjacent cycle nodes and also all cycle nodes with the depot. The edges representing adjacent cycle nodes have zero cost and the edges to the depot represent the shortest path costs from the depot to the various cycles on G_2.

Choosing a different, but specific, set of cycles yields an interesting result. The set of cycles which were chosen represent a checkerboard pattern. Let G_3' be the graph of cycle nodes obtained by choosing the opposite checkerboard (the white blocks rather than black). G_3' is then the dual graph of G_3. (A dual graph is one with nodes representing the spaces in the original graph and edges

joining nodes which represent adjacent spaces.)

If the contiguous district constraint is relaxed, a complete graph of cycle nodes could be constructed. Since, for contiguous districts, the serviced edges must be connected, only edges between adjacent cycle nodes were considered on G_3. These edges had zero cost, equal to the shortest distance between the closest nodes of pairs of cycles. Non-adjacent cycle nodes could be combined if districts did not have to be contiguous. An edge would exist between all pairs of cycle nodes with a cost equal to the shortest path between cycles on G_2. The shortest path between cycles is determined by finding the shortest path from each node in one cycle to all nodes in the other cycle. The shortest of these is the shortest path between cycles.

3.3.4. Capacitated minimum spanning tree

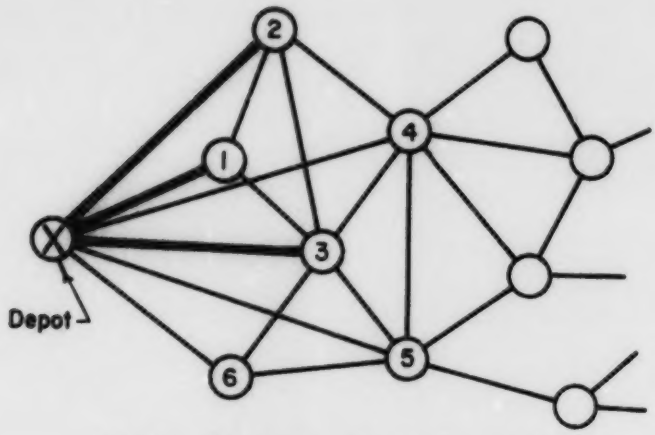
The objective of the overall problem is to minimize retraced distance. On G_3, all retraced distances, ignoring duplicated edges, are represented by the edges from the depot to each cycle node. The objective then becomes to minimize the total distance of these edges used in the solution. Since m tours are desired, the final solution will have m edges, called arms, leading from the depot. Each arm represents the path traveled from the depot to the district and back again after the district has been serviced.

Each district is represented by a subtree of cycle nodes built on each arm. A feasible solution has subtrees with loads less than capacity. The objective is to find the least cost set of m arms such that feasible subtrees can be built. This tree, including the arms, is the capacitated minimum spanning tree. If the minimum set of arms is found, only a feasible set of subtrees need be determined to guarantee an optimal capacitated spanning tree, since all edges other than the arms have zero cost and therefore do not add any cost to the total solution.

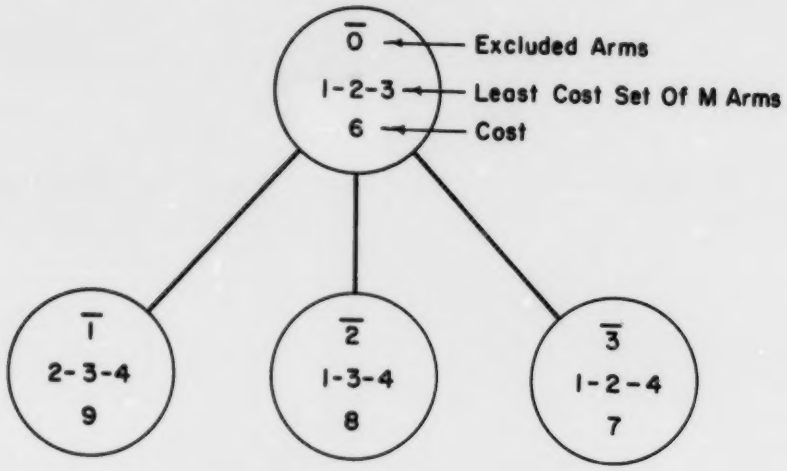
3.3.4.1. Branching procedure to find optimal arms

Since a feasible tree cannot be guaranteed for the least cost set of arms it is necessary to list possible sets in order of increasing cost. At each iteration the next least costly set is tested when the previous set does not provide feasible districts.

The method uses a decision tree with each node representing a set of arms and its associated cost. Figures 3b, 4b, and 5b represent the growth of the decision tree for a problem requiring three arms. Assume the cost of each arm equals the arm number. The first node of the decision tree is simply the m least cost edges leading from the depot. Figure 3a shows a portion of the cycle node network with heavier lines representing the three chosen arms. The top decision node in Figure 3b illustrates the combination of



(a) Network Of Cycle Nodes



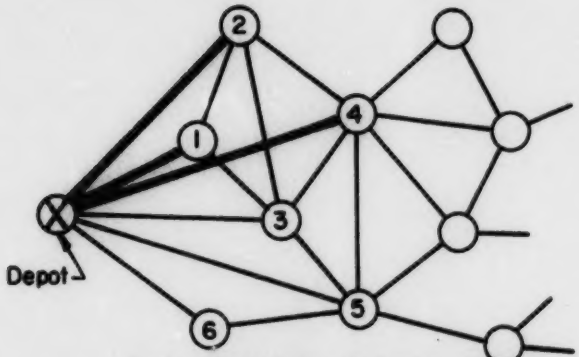
(b) Decision Tree With Decision Node Representing Arms 1, 2, 3 And 3 Branches From The Decision Node

FIGURE 3. CYCLE NODE NETWORK AND DECISION TREE SHOWING 1-2-3 SET OF ARMS

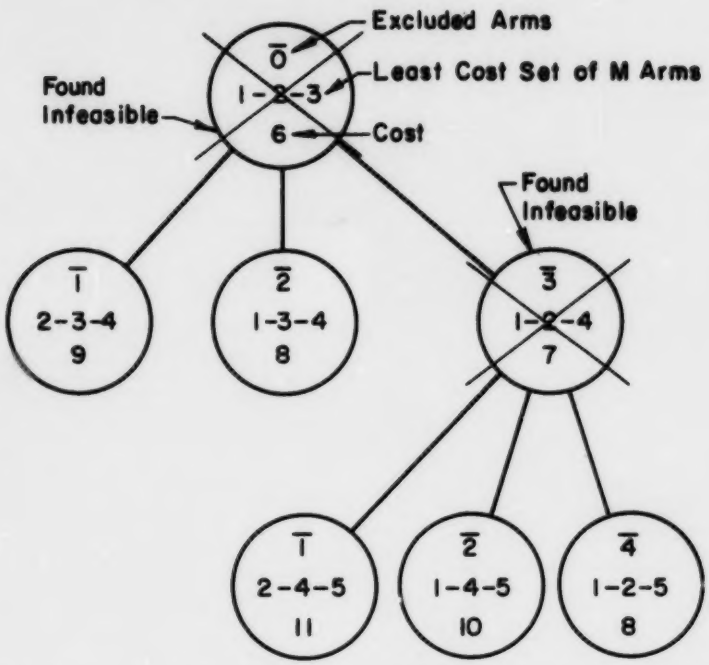
arms on the decision tree. Feasible districts are attempted using these arms by building a subtree of cycle nodes on each arm. Sections 3.3.4.2 and 3.3.4.3 explain the details of this operation. If feasible districts can be built, the branching algorithm is bypassed. If not, m branches are formed from the node, each representing the elimination of one of the m arms from the previous set. These branches are shown in Figure 3b. A new minimum cost set of arms is then chosen for each branch without using the excluded arm. These included arms are the m least cost non-excluded edges leading from the depot.

The m sets are then listed in order of ascending cost and the least cost combination is used to try to build feasible districts. If no districts are possible the least cost set of arms is eliminated from the list and a branch is made on the decision tree from the node representing that set. In Figure 3b the 1-2-4 set of arms represent the least cost set. Figure 4a shows these arms on the network. Assuming the arms are infeasible a branch is made on the decision node representing the 1-2-4 set of arms. That branch is shown in Figure 4b.

Branching is done as before, eliminating one arm for each branch. The sets of arms on the resulting branches now have two arms excluded from consideration, one from the first branch and the appropriate arm from the most recently



(a) Network of Cycle Nodes



(b) Extended Decision Tree With 3 Branches From the Decision Node Representing Arms 1-2-4

FIGURE 4. CYCLE NODE NETWORK AND DECISION TREE SHOWING 1-2-4 SET OF ARMS

attempted set of arms. The sets just obtained are inserted into the list such that the order of ascending cost is maintained. Again the least cost set of arms is used to attempt to build feasible districts. If none is found, that set is eliminated from the list and the decision tree branches on the node representing the most recently attempted set. In the example there is a tie for the least cost set of edges. Figure 4b shows the two tied sets of arms (sets 1-3-4 and 1-2-5) each with a cost of 8. It is immaterial which set is tried first since they both have equal cost. The algorithm simply chooses the next set on the ascending cost list of sets. In the example the 1-3-4 set is used. These arms are shown on the network in Figure 5a. Again assuming that the 1-3-4 set is found infeasible, branches are made from the 1-3-4 decision node. The m branches are shown in Figure 5b. The procedure continues until a feasible set of arms is found and then the remainder of the m postman's algorithm is completed.

Each time the procedure branches it is not necessary to branch on all m arms. As an example, assume the arms are numbered in increasing order of cost and also assume arm number 3 was excluded from the set from which the branch is being made. It is necessary to branch only on the arms in the set which are greater than 3. This eliminates duplication. Another branch was made previously on arm number 1.

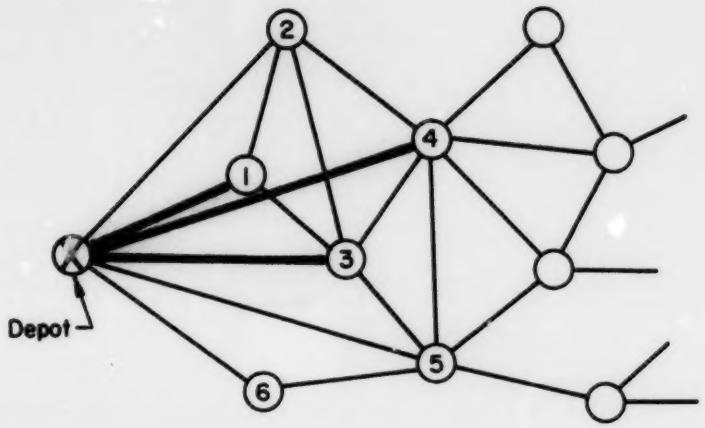


FIGURE 5a. CYCLE NODE NETWORK SHOWING 1-3-4 SET OF ARMS

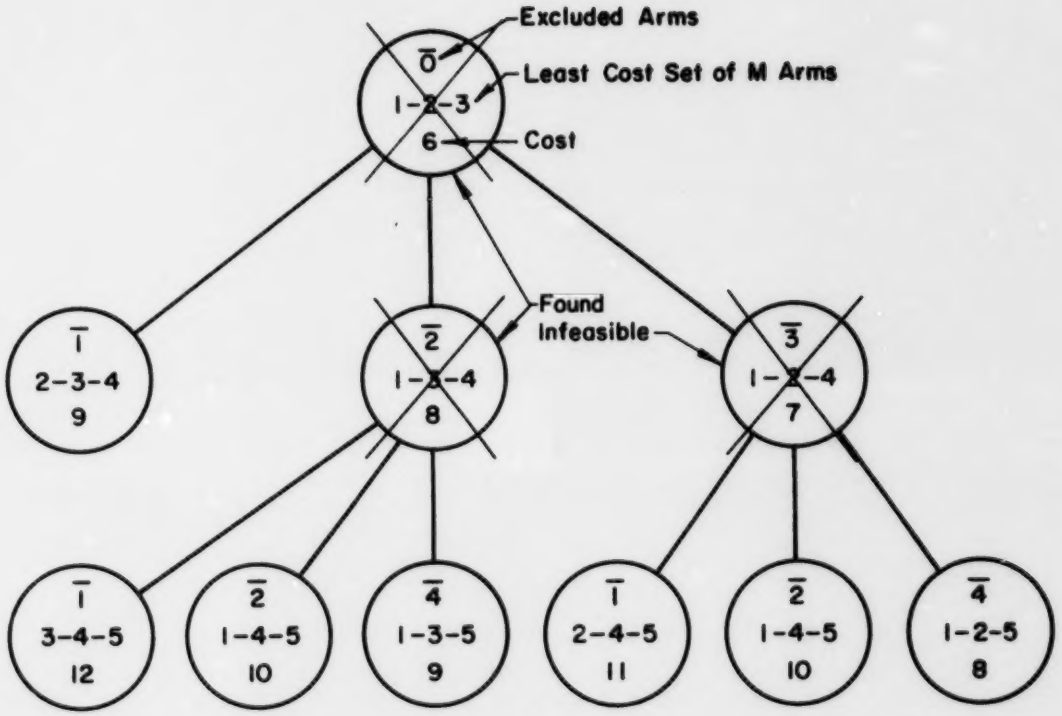


FIGURE 5b. EXTENDED DECISION TREE WITH 3 BRANCHES FROM THE DICISION NODE REPRESENTING ARMS 1-3-4

One of those branches will eventually be on number 3. Therefore a branch of the decision tree represents the least cost set of arms with both number 1 and number 3 excluded. There is therefore no need to branch on number 3 to number 1 and duplicate the same least cost set. This reasoning is simply explained by the fact that the code is concerned only with combinations and not permutations. Figure 6 shows a totally enumerated tree with three arms being chosen from a possible set of six. Assume that the cost of each arm is the same as the arm number. The nodes in the tree in Figure 6 represent all 20 possible combinations of 6 things taken 3 at a time.

It is never necessary to branch more than once at each iteration of the branching routine. Any branch beyond nodes which still might be feasible is unnecessary, since the costs of these branches must be greater than that on the node from which the branch was made. Since only one combination of nodes is found infeasible at each iteration, branches need to be made only on this decision node. When one set of arms is found infeasible, a branch must be made from the node representing that set. A branch from this set may yield a set slightly more expensive than the preceeding set but less costly than branches made simultaneously with the preceeding set.

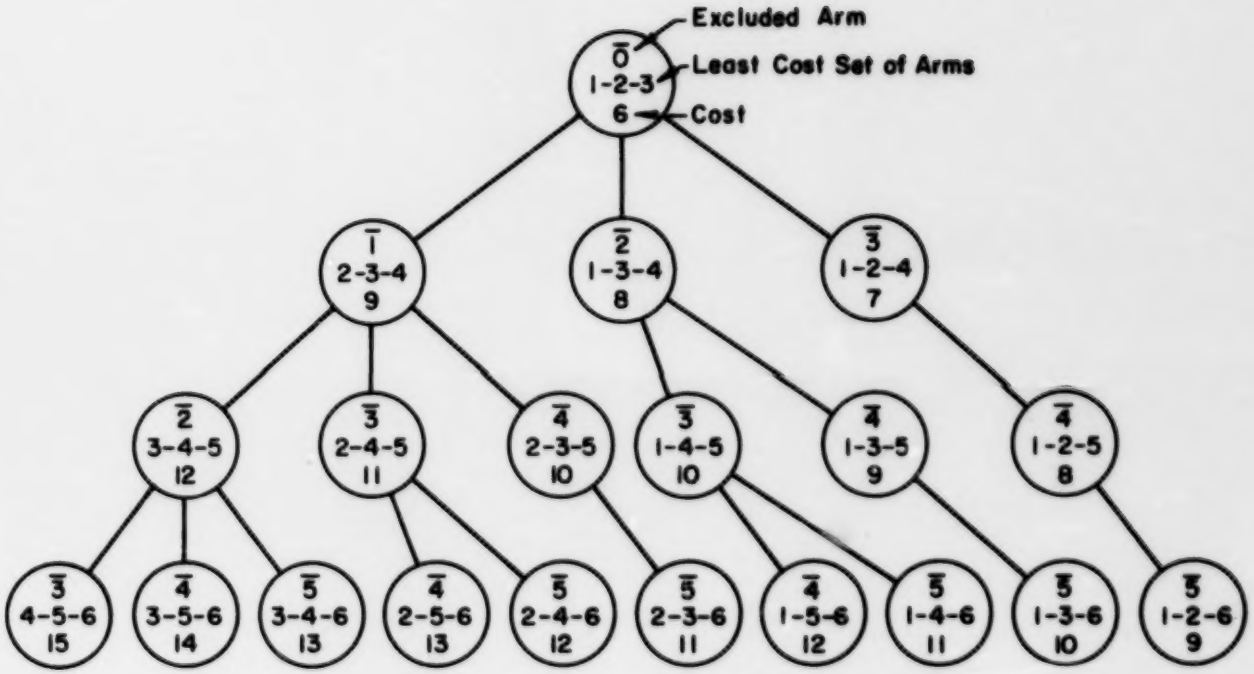


FIGURE 6. ENTIRE DECISION TREE FOR 6 NODES TAKEN 3 AT A TIME

3.3.4.2. Feasibility criterion for arms

Several techniques are used before and during the tree building section to determine if the selected arms can yield feasible districts. The screening process incorporates three features. First, all cycle nodes must be within reach of at least one arm. A cycle node is within reach if a tour can start at an arm, collect to the cycle node in question, collect the load on that cycle node, and collect on the way back to the arm. This sequence simply represents a path of cycle nodes connecting the arm and the cycle node. Paths exist for all cycle nodes, but the total load on the included cycle nodes might exceed capacity. If no feasible path exists the set of arms is infeasible because no capacitated tree, rooted at any arm, can be built which includes the unreachable cycle node.

The second screening feature deals with isolated groups of cycle nodes. It is possible for two or more arm nodes to isolate another arm. In other words, the only edges leading from one arm go directly into other arms, making it impossible to add any cycle nodes to that arm. This situation could also arise with a small group of cycle nodes surrounding an arm. It still might be possible for the remaining arms to have feasible subtrees. If the remaining m - 1 trucks, with capacity C can collect m times the average load, A, then one arm (truck) need not be used

at all. If x is the load on an isolated arm or small group of cycle nodes, then m*A-x is the remaining load to be collected by m-1 trucks. Therefore, C*(m-1) must be greater than m*A-x. Simple calculations show that x must be greater than c-m(C-A) to obtain feasible districts.

The third test applies after portions of the tree have been built. It might be necessary for certain cycle nodes to be attached to one arm. The situation would occur if a cycle node can be reached by only one arm. A path from the cycle node to that arm must, therefore, be part of the subtree attached to that arm. If the load on the resulting path is close to capacity then other cycle nodes might be stranded, in that no path exists from them to any arm without passing through the existing district. The stranded cycle nodes, therefore, cannot be attached to any arm and the set of arms is infeasible.

These three tests of feasibility are applied each time a new set of arms is chosen and, when necessary, during construction of the tree.

3.3.4.3. Building subtrees

Once arms have been chosen and not determined infeasible, subtrees are constructed to represent the m districts. In an attempt to keep the district loads balanced, adjacent cycle nodes are added sequentially to the district (subtree) with the smallest load. One cycle node

at a time is added until all are included in a subtree.

Before this procedure is performed it is necessary to determine if a cycle node or group of cycle nodes need to be attached to a certain arm. This would be true if only one arm can reach a certain cycle node. If so, the cycle node in question and a path from this cycle node to the arm have to be in the capacitated spanning tree solution. If the path chosen to reach the distant cycle node creates an infeasible situation then all paths which might eliminate the possibility are tried in an attempt to find a feasible configuration. By including this path before building the tree, infeasible sets of arms are more quickly discovered.

3.3.4.4. Balancing district loads

After a spanning tree has been built it may still violate the capacity requirements. One or more subtrees, or districts, could have a load greater than capacity. The districts might possibly be balanced by exchanging cycle nodes from one district to another. The exchanged cycle node, of course, must be adjacent to both districts in order to satisfy contiguity. Also, the district from which a cycle node is removed must remain connected. If removal of a cycle node divides the district into two disjoint sets, the trade is not allowed.

Two criteria are used to improve the balance of the districts and reduce the load on violating districts to less

than capacity. The first evaluates the change in district load relative to the average district load. Let D_1 and D_2 be the loads on two adjacent districts and A be the average district load. If $|D_1 - A| + |D_2 - A|$ can be decreased by exchanging a cycle node from one district to another then the change is made and the district loads are recalculated accordingly. Note that the evaluator will decrease only for districts where the load in one is greater than A and the other less. This criterion is applied until either all districts are feasible or no further improvements can be made.

The second criterion is used only after the first fails to find feasible districts. It enables an exchange between any two adjacent districts regardless of their size relative to A. If $|D_1 - D_2|$ can be reduced by moving a cycle node then the trade is made. In both criteria, changes are made only when the values are decreased. Therefore, the procedure converges until no more changes are feasible.

If the above procedure leads to a feasible solution for a given set of arms, the algorithm terminates. If a feasible solution is not found, the search begins again on the next set of arms. It should be noted that a feasible solution may exist for some set of arms and yet not be found by this procedure. Thus, the algorithm is heuristic and not necessarily optimal.

CHAPTER FOUR

HEURISTIC ALGORITHM

4.1 Detailed outline

Step 1

If the network has no odd nodes go to step 2; otherwise create a unicursal graph by solving the CPP.

Step 2

Establish cycles on the unicursal graph.

Step 3

Create a graph of cycle nodes.

Step 4

Find the least cost set of arms not yet found infeasible.

Step 5

Screen the arms to determine if they are infeasible. If infeasible go to step 4. Otherwise go to step 6.

Step 6

Add cycle nodes to the arms to build subtrees.

If during construction, the tree is found infeasible, go to step 4.

If the districts are feasible (including load balance) go to step 9. Otherwise go to step 7.

Step 7

Exchange cycle nodes to balance the districts using criterion 1.

If the districts are feasible and balanced, go to step 9.

If step 8 has been performed once and no changes were subsequently made in step 7 go to step 4.

Otherwise go to step 8.

Step 8

Exchange cycle nodes using criterion 2.

If the districts are feasible and balanced go to step 9.

If a change was made in step 8 go to step 7.

Otherwise go to step 4.

Step 9

Combine cycles which are connected as cycle nodes in the capacitated spanning tree.

For any sequence of traveled edges make sure the path used is the minimum cost path. If not find the minimum cost path and proceed.

List the edges in the m tours.

4.2. Lack of a guarantee of optimality

The preceeding algorithm is a heuristic because it cannot guarantee an optimal solution. Four reasons for this are discussed in this section.

The first deals with establishing the cycles on the unicursal graph. Once edges are combined to form a cycle they remain together throughout the algorithm and will be in the same district in the final solution. There is no guarantee that the checkerboard set of cycles is necessarily the optimal configuration and that the edges in one cycle should be in the same district. The checkerboard set is chosen because it is the most easily manipulated. It also represents the most likely intuitive grouping of the edges, in that a feasible solution will have tours made up of cycles with these characteristics.

The duplicated edges, which were added to create a unicursal graph, represent the second flaw in the procedure. These edges require service even though their load is zero. However, their addition to the network is necessary to create the cycles. Although the duplicated edges minimize additional cost they are ignored during the subsequent minimization of arm lengths. Consideration of the duplicated edges might alter the choice of paths between the depot and various districts. The situation is best illustrated with an example. Figure 7a shows a section of a network with two retraced edges (dashed lines). The numbers next to the edges represent the appropriate costs and show that the shortest path is duplicated. Assume the edges are all in one district and that the arm connecting them with the depot is

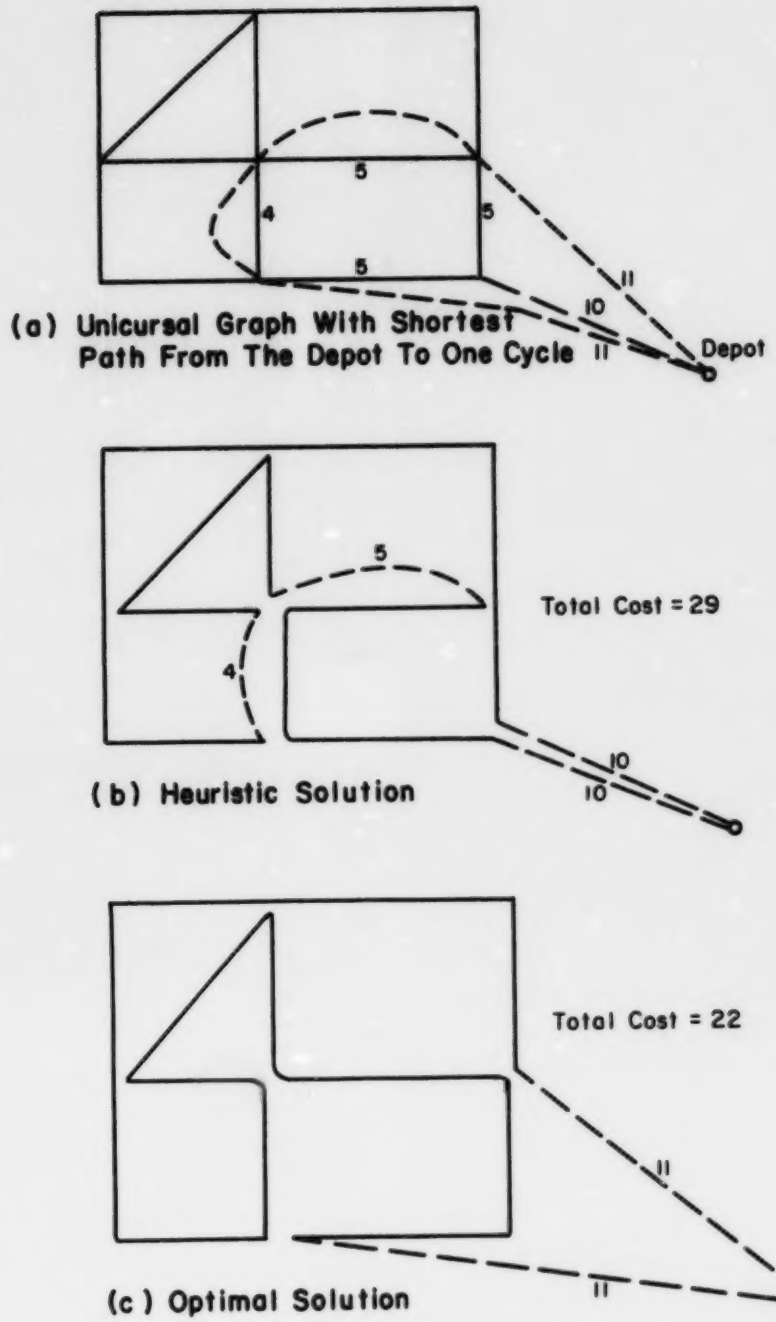


FIGURE 7. EXAMPLE OF THE HEURISTIC AND OPTIMAL SOLUTION ON A SMALL NETWORK

attached to the lower right-most node. Let the cost of the arm be 10. Also assume the cost of the path from the depot to each odd node is 11. The tour generated by the algorithm is shown in Figure 7b with a total retraced distance of 29. However, a lower cost solution of 22 is found by using the two odd nodes as the entry and departure points and eliminating the duplicate edges. The tour is illustrated in Figure 7c. This example is simplified, but represents what might occur in a solution.

The third reason which prohibits an optimal guarantee is that the capacitated spanning tree may not be optimal. A feasible capacitated spanning tree may exist on a certain set of arms but not be found by the tree building heuristic. When a feasible capacitated spanning tree is not found, even though it may exist, the algorithm proceeds to another combination of arms. This combination is most likely more costly and therefore not optimal.

The fourth counterexample is associated with the contiguous district constraint. The definition of a contiguous district is that all serviced edges in one district form a connected graph. In rare instances, the algorithm may terminate with a noncontiguous district. Although the algorithm requires duplicated edges to be serviced, they are actually traversed, since their loads are zero. The situation arises when two edges in one cycle are duplicated

edges. Assume these two traversed edges are on opposite sides of a cycle with four edges. The two remaining serviced edges are therefore not connected by a serviced edge. Other serviced edges in other cycles will be connected to one of the two serviced edges. However, there are no serviced edges connecting the two groups, creating a noncontiguous district. Since the sum of retraced districts is minimized, the retraced edges in question have a low cost. Therefore, the separated sections of the district will be relatively close.

4.3. Notes on the heuristic

Several observations should be made about the heuristic procedure which shed light on its operation. To obtain a feasible solution, vehicle capacity, C, must be greater than the average load per truck, A. The larger the difference between the two, the more flexible is the algorithm. It is possible that no feasible solution exists even though C is larger than A. In most cases, though, as the value C-A decreases, the algorithm will find a solution, but a more costly one, simply because the constraints are more confining. Of course several other factors, including m, network configuration and relative sizes of cycle node loads, influence the outcome. But in general the difference between capacity and average load has the greatest effect on the cost of the solution.

When calculating the arms to be used in obtaining a feasible solution, no guarantee exists that the optimal capacitated minimum spanning tree will result from a certain set of arms. However, the subsequent differences in costs among possible combinations of arms are generally very small. The algorithm then could by-pass the optimal capacitated spanning tree and terminate with a solution which is not far from optimal.

The problem is defined on an undirected network, but feasible solutions might include directed edges. Two different possibilities occur. The first deals with the calculations of the shortest paths from the depot to all cycle nodes. If directed edges with no load occur in the network, they would not be included in the cycles but could be used for traveling. Therefore, they might be considered when finding shortest paths from the depot to the cycle nodes. Rather than doubling the undirected shortest path, two directed shortest paths are calculated; one from the cycle node to the depot and the other from the depot to the cycle node. The cost is simply the sum of the two directed costs and the pair of paths is considered as one arm until the tree is decoded.

Directed edges with a load greater than zero might be allowed in the network itself, under certain circumstances. The individual cycles are traveled in only one direction in

the final solution. If directed edges are included in the cycles and they are all directed in the same way, then the results are still valid. Obviously, if only one directed edge appears in a cycle the statement is also true. Therefore, it is likely that a network with a small number of directed edges will have a feasible solution using the algorithm.

The process of building the capacitated spanning tree and then exchanging cycle nodes can easily be seen on G_2, the unicursal graph. District boundaries can arbitrarily be drawn on the network with one rule in mind; always divide nodes such that each district has an even number of edges incident to that node. Districting in this manner is feasible since all nodes have an even number of incident edges. District boundaries can then be moved to balance the loads. While moving the boundaries, however, each node must remain even within each district. The resulting changes are actually the moving of one or more cycles from one district to another, and correspond to the exchanging of cycle nodes in the capacitated spanning tree.

CHAPTER FIVE

RESULTS AND DISCUSSION

5.1. Computational results

The algorithm was coded in Fortran IV and implemented on an IBM 360/75 computer, using 348 kilobytes of storage. A computer listing, specific details, and an example problem appear in an appendix to this report. Twelve examples were tested and the results are shown in Table 1. The example networks were constructed to simulate real world situations. The edge costs were arbitrarily chosen within a certain range so that they were relatively realistic. Arbitrary values were also used for loads and consequently average loads and capacities have no real world meaning, except as relative to each other.

The solution times in Table 1 include the time required to solve the matching algorithm on the original network, create the cycles, form G_3, solve for the capacitated spanning tree, and decode the solution into the tours.

Several factors affect the efficiency and results of the algorithm. As stated earlier, truck capacity must be considerably larger than the average truck load. The capacity used in the examples ranged from around 15 to 25 percent larger than the average load. This range proved to be adequate to allow meaningful solutions.

Several examples terminated quickly when only a few

Problem No.	No. of Nodes	No. of Edges	No. of Districts	Capacity	Ave. Load	No. of Edges Incident To Depot	Solution Time	
							min.	sec.
1	21	37	4	78.	62.6	2	0	55
2	40	71	5	180.	138.	2	1	02
3	40	71	5	190.	138.	3	0	17
4	42	73	5	170.	143.	3	0	47
5	60	103	4	470.	372.	3	1	10
6	60	103	5	375.	298.	3	0	17
7	80	147	5	510.	431.	5	0	33
8	80	147	6	430.	359.	3	5	31
9	83	153	5	330.	278.	3	0	52
10	83	153	5	345.	316.	4	2	08
11	83	153	6	275.	231.	4	0	43
12	83	153	6	285.	265.	5	3	59

TABLE 1. COMPUTATIONAL RESULTS

combinations of arms were tried before a set was found which allowed feasible districts. Computation time increased considerably as more combinations of arms were attempted. The number of attempted arms depended on m and the configuration of the network. Under certain circumstances these two factors create a bottleneck in the network. A bottleneck is most easily defined in terms of a cut, which is a partitioning of the network (G_3) into two groups. All zero cost edges are in one group or the other. The cut is therefore a dividing line containing one or more cycle nodes but no edges. Any number of cuts can be designated on the network, as long as the network is divided into two groups. A cut becomes a bottleneck in the following situation. Assume that all the arms are chosen as cycle nodes on the near side of the cut (the side with the depot). Also assume that t cycle nodes make up the cut. If the load beyond the cut is greater than t times the capacity, then the cut is a bottleneck. The load beyond the cut can not be collected because only t subtrees, each attached to an arm, can be built past the bottleneck. Therefore, no feasible tree will be found and a new set of arms will be attempted.

To eliminate the bottleneck one or more cycle nodes, beyond the cut, must be chosen as arms. Since the arms are chosen in a sequentially increasing order any cycle nodes

which are distant from the depot, as these would be, would not be included until after considerable computation time. The sequence of choosing arms, screening for infeasible combinations, and attempting to find a feasible tree proves to be quite time consuming. When a bottleneck occurs many combinations of arms attempted before a feasible set is found.

The number of edges incident with the depot has considerable bearing on the existence of bottlenecks. A more centrally located depot is more likely to have a greater number of incident edges and therefore a lower probability of creating a bottleneck.

Most of the networks were too large to determine if the solutions were the absolute minimum cost solution. However, of the 12 examples run, 8 terminated with the optimal capacitated spanning tree. In other words, a capacitated spanning tree was found for the least cost feasible set of arms in eight instances.

5.2. Practical potential

This research is an attempt to provide a useful procedure which can be implemented on an actual street network. The resulting algorithm is useful not only when implemented as a computer code but also as a means of more easily visualizing and solving the problem by hand. The algorithm simplifies the manipulation of districts by

creating a graph of cycle nodes. This reduces the number of variables since each cycle node contains information for several edges of the original network. Reducing the network size in this way allows the user to more easily manipulate the cycles and study the various alternatives. Consequently, solving for a capacitated spanning tree on the cycle node graph is far easier than determining feasible tours on the original network. The size of the cycle node graph could be further reduced by combining pairs or groups of nodes before solving for the capacitated spanning tree.

The algorithm requires that capacity be somewhat larger than the average truckload. This requirement is realistic since the capacity of a truck is flexible and will allow compaction to the point where the amount of solid waste loaded into a truck exceeds its listed capacity. This flexibility allows a value larger than capacity to be used in the algorithm. The algorithm frequently terminated with considerable leeway between district loads and capacity. However, the final result does not reflect the need for a 15 to 25 percent difference between the average load and capacity in the algorithm itself. Using a larger value of capacity permits the algorithm more flexibility in finding feasible districts when exchanging cycle nodes between districts. If the algorithm should terminate with a district close to or equal to the larger capacity, the listed capacity

might be violated but the tour is still valid because of the flexibility of the truck.

One possibility which might hinder the use of the algorithm is the existence of a large block, such as a park, in the network. If the streets surrounding the block have a load greater than capacity then one truck can not collect around the entire cycle. If this situation occurs and the block is designated as a cycle then duplicate edges would have to be added to create two or more cycles from the large one, each with a load less than capacity.

5.3. Implementation

As presently coded the algorithm can solve the MPP on a network with up to 90 nodes and 270 edges. As many as 46 nodes in the original network may be odd. Wathne (33) studied a 0.2 square mile residential area in Baltimore, Maryland with a population of 4,183. According to Truitt et al (32), Baltimore's residential solid waste generation rate is 1.95 lbs/cap/day. The street network in the area has 46 nodes and 75 edges. The standard size truck, as used in Baltimore, can hold about 10,000 lbs of solid waste. Therefore, three truckloads are needed to collect the area. Extrapolating the network size, shows that a network of 90 nodes would need to be partitioned into about 6 districts.

A large network could be handled by the computer code if the code were more efficiently written and if more

computer core was used for storage. However, computation time might increase considerably as the size of the network is increased. This statement is supported by the fact that bottlenecks are much more likely to occur as m is increased. As the network is increased in size, m would have to be increased proportionately in order to service the network. However, for any network the configuration around the depot is fairly uniform. Rarely would more than six edges be incident to the depot. Therefore, as the network size and m increase, bottlenecks occur more frequently involving more computer time.

The size restriction on the code does not seriously limit the use of the code on a large network. Currently, planners usually divide the city into smaller collection areas which can more conveniently be subdivided into districts. Choice of collection areas is aided by natural boundaries within the city such as rivers, railroads, interstate routes, and parks. The MPP algorithm is useful in determining districts within these areas. If transfer stations are used then the areas might be decided with them in mind and the station would be a local depot. However, if the transfer station or disposal site is remote and not part of the area under consideration, then one node would be designated as an artificial local depot for that collection area. The node closest to the remote depot should be chosen

in order to minimize the distance between the two.

A district was defined as the streets serviced by one truck in one tour. One truck can generally service more than one district in a day and it is necessary to assign two or more districts to each truck. Two alternatives might be considered. The districts assigned one truck might be adjacent and therefore each truck would service one contiguous sector during a single day. Another method might attempt to balance the times traveled by the trucks during the day. If one truck services a district far from the depot, it would also service a close district. At the same time a second truck would service two districts at an intermediate distance from the depot.

Underlying the algorithm is the assumption that one district can be serviced in eight hours. Servicing more than one district by one truck, however, might require more than eight hours. This constraint should be kept in mind when assigning districts to trucks.

The algorithm determines tours as the districts are formed. However, it may be desirable to alter the tours given in the solution. The edges in each tour form a unicursal graph since it is a cycle. Therefore, any of the tour building algorithms can be used on the unicursal graph formed from the edges of each tour. Liebman (25) has developed a scheme to eliminate all u-turns from the tour on an undirected

unicursal graph with no dead-end streets. The procedure changes the travel direction on certain cycles of the tour to eliminate any existing u-turns. Liebman's scheme, as well as the others, could be added as a final step of the program.

Computer implementation of the algorithm has the advantage of rapid solution time. Its usefulness is greatly felt when changes occur in the street network. New streets, urban renewal, population changes, and changes in traffic patterns would all effect the routes used by collection vehicles. If changes occur in the network the data can easily be changed to conform to the new situation and the algorithm can be resolved. Modified tours can then be implemented to handle the change in real world conditions. This procedure can be repeated, with little difficulty, whenever warranted.

The solution to the algorithm is not necessarily the plan which should be implemented. The solution merely provides information for the system manager. He may decide to adopt or reject the plan depending upon the solution characteristics. He may also alter the solution to more closely meet his own specifications. A lower cost solution may be apparent after a close inspection, or more compact, but still feasible, districts might be obtained by visually moving district boundaries.

The algorithm does provide a basis from which the user may operate. Knowing the objective, constraints, and limitations of the algorithm he can easily adopt, reject, or modify the solution to better serve his needs.

5.4. Possible modifications of the algorithm

Many changes could be made in the algorithm which would alter either the objective or some of the constraints. Several alterations have already been mentioned. Non-compact districts could be considered by constructing a complete graph of cycle nodes. On this network a capacitated spanning tree would be constructed to determine feasible districts. However, a new routine would have to be developed to find the capacitated minimum spanning tree, because the present tree building routine takes advantage of the lack of non-zero cost edges in the network. Several schemes were tried on the complete cycle node network, but were found far inferior to the present procedure.

The procedure of choosing m arms and then building capacitated subtrees is not necessarily the best method for determining a low cost capacitated spanning tree. An optimal scheme or faster non-optimal procedure might be developed which could easily be inserted into the algorithm.

The algorithm might also be modified to consider more than one depot. If the network has two depots the algorithm would simply build two capacitated spanning trees.

The number of tours incident with each depot might be specified or variable.

Directed streets can be handled by the algorithm if each designated cycle has streets directed in the same way. If this requirement is not met then the algorithm could be changed to try and find cycles which have properly directed edges. As long as the network is unicursal, cycles do exist which satisfy this criterion. However, the number, configuration, and load on the cycles may not be amenable to the MPP solution.

A slight modification will allow a remote depot to be considered, rather than requiring the depot to be a node in the collection area. The arms would again be chosen from the shortest paths from the depot to each cycle node.

5.5. Extensions and other applications

The MPP has application in other real world problems and might prove to be useful in these areas. The mere name of the problem suggests that the results can be used to determine postal carrier's routes. Other public service vehicles which have very similar characteristics are snowplows, school buses, and street sweepers.

Slight modifications would be needed in some cases. Snowplows should be constrained to service major streets before lightly traveled ones. Major streets might also need to be plowed more than once. The same criterion may also be

applied to street sweepers. The school bus routing problem appears identical to the MPP and can utilize the same solution procedure.

CHAPTER SIX

SUMMARY AND CONCLUSIONS

The algorithm presented in this paper routes a known number of collection vehicles over all the edges of a connected, planar network. Each edge in the network has a load of solid waste which is collected during one of the tours, each of which begins and ends at a prespecified depot. The sum of the loads collected on one tour does not exceed a specified capacity and the collected edges in each tour form a contiguous district.

The creation of districts on the network is based on the existence of a unicursal graph. The original network does not necessarily possess this quality and therefore the unicursal graph is developed by solving the CPP algorithm on the network. To facilitate the designation of districts on the unicursal graph a new network is created which completely represents the unicursal graph. To create this network edges are grouped into cycles on the unicursal graph. A specified set of cycles is chosen because it represents the most likely combination of edges in view of the desired solution. On the new network, nodes represent the cycles on the unicursal graph and zero cost edges join cycle nodes which represent adjacent cycles. No edges exist between cycle nodes representing non-adjacent cycles. An artificial node is created on the new graph to represent the depot and

edges are added from the depot to all cycle nodes. Costs on these edges equal the costs of the shortest paths, on the unicursal graph, from the depot to the cycles.

To find districts, a capacitated spanning tree is built on the new network. The capacitated spanning tree is made up of m subtrees, each of which has exactly one edge incident with the depot and a number of cycle nodes connected to this edge, in the form of a tree. The subtrees represent the districts, so the sum of the cycle node loads on each one must not exceed capacity.

Because the subtrees (districts) are a combination of adjacent cycle nodes, the tours are created by simply combining the cycles on the unicursal graph, as they are joined in the spanning tree solution on the new graph. The edges in each tour will be a unicursal graph because the tours are comprised of cycles. The edge from the depot to each district is doubled to complete the tour.

The procedure differs from most, in that the districts and tours are determined simultaneously. Most previous algorithms first found logical districts and then solved the CPP on each district to determine its tour. Of course, in many different cases, different criteria were used to define the problem and therefore comparing solutions is inappropriate. It would be a problem in itself to determine the best measure of effectiveness for solid waste collection.

Consideration of all factors, such as travel time, travel distance, left hand turns, u-turns, district contiguity or compactness, and variation in crew work loads would be most desirable. The problem is how to compare the various factors and determine their relative value. The algorithm presented in this paper minimizes the cost associated with operating a solid waste collection vehicle. This cost is not only the largest factor in the cost of solid waste management but also the factor which can provide the greatest savings. Other less significant variables are changed to improve the solution characteristics after the main criterion has been minimized.

The algorithm was developed to represent real world situations as closely as possible. Of course simplifying assumptions must always be made to enable an efficient solution procedure. The most limiting factors of the algorithm are the restrictions on directed streets and the size limitation of the program. These do not present serious limitations, however. A large network can easily be subdivided into areas capable of being handled by the algorithm. Directed streets can be included if their configuration is feasible relative to the derived tours.

It is impossible to incorporate many real world constraints into the procedure. Human nature is difficult to anticipate and incorporate into any meaningful operating

procedure. For example, each crew may have different preferences as to routes, operating procedures, and lunch and coffee breaks. Efficient solutions may go counter to their experience or hunches and present an impasse to an efficient solution. Explaining the reason for the tour configuration to the collection crews might help gain its acceptance at that level of operation.

The lack of a guarantee of optimality does not significantly reduce the impact of the solution. The results are meant to be a source of information and not necessarily the final say in efficient tours. As a source of information the solution can be studied and altered in various ways. An optimal solution should also be a source of information and be used in the same manner. An optimal solution does not offer much more to the user than the knowledge that it is the least cost solution given the operating constraints.

Apart from being a potentially useful algorithm it is hoped that the theory behind the procedure will encourage further research in the field. The algorithm is by no means the final step in the study of solid waste collection. The study of operations research applied to solid waste collection is relatively recent. Over one half of the publications in the field have appeared since 1970. This fact emphasizes the growing concern for an importance of one of the most

costly municipal management problems. This research is intended to provide insight for immediate improvement in solid waste collection practices and at the same time establish grounds for future research.

APPENDIX

A. LISTING OF COMPUTER CODE

MAIN

```
0001            COMMON DIST(90,90),IPATH(90,90),JPATH(90),IEND(270),
                1JEND(270),COST(270),NODD(90),ELOAD(270),NSOURC,NTERM,INUM,
                2 NUMNDS,NUMEGS,IINODE(1080),JJNODE(1080),AROUND(90,10),NUMINC(90),
                3 INNODE(60,10),CYCNDS(60),SP(60,60),NRSTND(60,60),CLOAD(60),
                4 DLOAD(20),IN(20,40),NODE(90),NO(20),JX(60,60),ITIE(100),NF(100),
                5 SPB(60),M,N,TCOST,INF,FINF,NCYNDS,IDEPOT,AVLOAD,C,NATCH,
                6 NEXTND(10),ITOUR(20,300),NOTOUR(20) ,STOP,IMOUT,COSTK(20)
0002            COMMON/KMOTPT/IKM(1080),JKM(1080),NONDS,NOEDGS, MATCH(1080),
                1 EDGWT(1080)
0003            COMMON/BILD/INTREE (50,20),IRETRN,NUMBER,IRMS ,HBEGIN,IBEGIN,
                1 IONE,ALIST(200),NEXT(200),IPACK(200,2), MASK(30),INJX(60),IDONE
0004            COMMON/ STOREE / STORE(90,90)
0005            COMMON /EPSIL/ EPSILN ,IFLAG
0006            INTEGER HBEGIN
0007            INTEGER AROUND,CYCNDS
0008            CALL SEC1
0009            IF(ISTOP .LT. 0.) GO TO 110
0010            CALL SEC2
0011            CALL SEC3
0012            IF(IMOUT .NE. 1) GO TO 100
0013            WRITE(6,350)
0014          350 FORMAT(//10X, 'LIST OF ATTEMPTED ARMS'//)
0015          100 CALL SEC4
0016            IF(ISTOP .LT. 0.) GO TO 110
0017            CALL SEC5
0018            IF(IRETRN .EQ. 1) GO TO 100
0019            IF(IMOUT .NE. 1) GO TO 120
0020            CALL OUT
0021          120 CALL SEC6
0022            CALL SEC7
0023          110 STOP
0024            END
```

```
0001            SUBROUTINE SEC1
                C
                C          SECTION 1 READS INPUT AND PRINTS IT FOR REFERENCE
                C
0002            COMMON DIST(90,90),IPATH(90,90),JPATH(90),IEND(270),
                1JEND(270),COST(270),NODD(90),ELOAD(270),NSOURC,NTERM,INUM,
                2 NUMNDS,NUMEGS,IINODE(1080),JJNODE(1080),AROUND(90,10),NUMINC(90),
                3 INNODE(60,10),CYCNDS(60),SP(60,60),NRSTND(60,60),CLOAD(60),
                4 DLOAD(20),IN(20,40),NODE(90),NO(20),JX(60,60),ITIE(100),NF(100),
                5 SPB(60),M,N,TCOST,INF,FINF,NCYNDS,IDEPOT,AVLOAD,C,NATCH,
                6 NEXTND(10),ITOUR(20,300),NOTOUR(20) ,STOP,IMOUT,COSTK(20)
0003            COMMON/KMOTPT/IKM(1080),JKM(1080),NONDS,NOEDGS, MATCH(1080),
                1 EDGWT(1080)
0004            COMMON/BILD/INTREE (50,20),IRETRN,NUMBER,IRMS ,HBEGIN,IBEGIN,
                1 IONE,ALIST(200),NEXT(200),IPACK(200,2), MASK(30),INJX(60),IDONE
0005            COMMON /EPSIL/ EPSILN ,IFLAG
0006            INTEGER AROUND, CYCNDS, HBEGIN
                C
                C          INPUT IS IN THE FOLLOWING FORM
                C
                C          **FIRST CARD-  NUMBER OF NODES IN NETWORK IN COLUMNS 1 -5,
                C            NUMBER OF EDGES IN NETWORK IN COLUMNS 6- 10, NUMBER OF
                C            DISTRICTS (TRUCKS) IN COLUMNS 11 - 15, NODE WHICH
                C            REPRESENTS THE DEPOT IN COLUMNS 16 - 20, TRUCK CAPACITY
```

```
C                IN COLUMNS 21 - 30 (REAL NUMBER) , AND A TERMINATION
C                COST, ABOVE WHICH WE DON'T WISH THE ALGORITHM TO
C                PROCEED, IN COLUMNS 31 - 40 (REAL NUMBER) ,    A ONE IN
C                COLUMN 45 IF INTERMEDIATE OUTPUT IS DESIRED (ZERO OTHER-
C                WISE), AND THE INCREASE OVER TRUCK CAPICITY BELOW
C                WHICH THE INFEASIBLE SOLN IS PRINTED (IF C +
C                EPSILN IS NOT VIOLATED AS CAPACITY THEN THE TEMP.
C                SOLN IS PRINTED) IN COLUMNS 46 - 54, A ONE IN COLUMN
C                60 IF CRITERION 2 IS TO BE USED IN SECTIONS.
C              ** NEXT CARDS (ONE FOR EACH EDGE) - THE TWO NODES INCIDENT
C                ON THE EDGE IN COLUMNS 1 -5 AND 6 - 10 (NODE NUMBERS
C                MUST NOT EXCEED THE NUMBER OF NODES), THE COST
C                ASSOCIATED WITH THE EDGE IN COLUMNS 11 - 20 (REAL
C                NUMBER), AND THE LOAD ASSOCIATED WITHTHE EDGE IN COLUMNS
C                21 - 30 (REAL NUMBER).
0007            INF = 9999999
0008            FINF = 999999.
C              NOTE - EDGE NUMBER ONE MUST BEINPUT SUCH THAT WHEN
C                GOING FROM THE FIRST INCIDENT NODE TO  THE SECOND THE
C                NETWORK IS ENTIRELY TO THE LEFT OF THE EDGE. ALSO EDGE
C                ONE MUST BE THE ONLY EDGE BETWEEN IEND(1) AND JEND(1),
C                (THE TWO NODES INCIDENT TO EDGE ONE)
C              ** NEXT CARDS - (ONE FOR EACH NODE) - THE NODES INCIDENT
C                TO THE RELEVENT NODE LISTED IN CLOCKWISE ORDER AROUND
C                THE RELEVENT NODE ,IN COLUMNS 1 -5, 6 - 10,11 - 15, ETC.
C              NOTE - WHEN LISTING THE NODES INCIDENT TO IEND(1),
C                (THE FIRST NODE INCIDENT TO EDGE ONE), WE MUST START
C                THE CLOCKWISE SEQUENCE WITH JEND(1), (THE SECOND NODE
C                INCIDENT TO EDGE ONE). THEREFORE AROUND(IEND(1),1) =
C                JEND(1)
C
C
C              READ INPUT
C
0009            READ (5,100) NUMNDS, NUMEGS, M, IDEPOT, C, STOP, IMOUT ,EPSILN,
               1 IFLAG
0010          100 FORMAT(4I5,2F10.0,4X,I1,F10.0, 4X,I1)
0011            DO 110 KK = 1,NUMEGS
0012          110 READ(5,120) IEND(KK), JEND(KK),COST(KK), ELOAD(KK)
0013          120 FORMAT(2I5,2F10.0)
0014            DO 130 I = 1,NUMNDS
0015          130 READ(5,140) (AROUND(I,J),J = 1,10)
0016          140 FORMAT(16I5)
C
C              CALCULATE NUMBER OF EDGES INCIDENT ON EACH NODE
C
0017            DO 180 I = 1, NUMNDS
0018            NUM = 0
0019            DO 170 J = 1,10
0020            IF(AROUND(I,J) .EQ. 0) GO TO 180
0021          170 NUM = NUM + 1
0022          180 NUMINC(I) = NUM
C
C              CHECK TO SEE IF DATA IS CONSISTENT
C
0023            IF (AROUND(IEND(1) ,1) .NE.  JEND(1)) GO TO 700
0024            ELOD = 0.
0025            DO 390 I = 1,NUMEGS
0026          390 ELOD = ELOD + ELOAD(I)
0027            IF(ELOD - M*C .GT. .00001) GO TO 700
0028            NUM = 0
0029            DO 400 I = 1 , NUMNDS
0030          400 NUM = NUM + NUMINC(I)
0031            NUM = NUM/2
0032            IF(NUM .NE. NUMEGS) GO TO 700
0033            DO 410 I = 1,NUMEGS
0034            IF (IEND(I) .GT. NUMNDS) GO TO 700
0035            IF(IEND(I) .LT. 1) GO TO 700
0036            IF(JEND(I) .GT. NUMNDS) GO TO 700
0037            IF(JEND(I) .LT. 1) GO TO 700
0038            IF (IEND(I)  .EQ. JEND(I)) GO TO 700
0039          410 CONTINUE
0040            DO 420 I = 1,NUMNDS
0041            NUMIN = NUMINC(I)
0042            DO 420 J = 1,NUMIN
0043            IF(AROUND (I,J) .GT. NUMNDS) GO TO 700
0044            IF(AROUND(I,J) .LT. 1) GO TO 700
0045          420 CONTINUE
0046            ELOD = 0.
0047            DO 430 I = 1,NUMEGS
0048          430 ELOD = ELOD + ELOAD(I)
0049            IF(ELOD .GT. M*C ) GO TO 700
```

```
0050              DO 440 I = 1,NUMNDS
0051              NUMIN  = NUMINC(I)
0052              KOUNT = 0
0053              DO 450 J = 1,NUMEGS
0054              IF(I .NE. IEND(J)) GO TO 450
0055              DO 470 JJ = 1,NUMIN
0056              IF(JEND(J) .NE. AROUND(I,JJ)) GO TO 470
0057              KOUNT = KOUNT + 1
0058              GO TO 460
0059            470 CONTINUE
0060            460 IF(KOUNT .EQ. NUMIN) GO TO 440
0061            450 CONTINUE
0062              DO 500 J = 1,NUMEGS
0063              IF (I .NE. JEND(J)) GO TO 500
0064              DO 490 JJ = 1,NUMIN
0065              IF(IEND(J) .NE. AROUND(I,JJ)) GO TO 490
0066              KOUNT = KOUNT + 1
0067              GO TO 500
0068            490 CONTINUE
0069            500 CONTINUE
0070              IF(KOUNT .NE. NUMIN) GO TO 700
0071            440 CONTINUE
0072              GO TO 185
0073            700 WRITE(6,690)
0074            690 FORMAT(///10X, 'ERROR MESSAGE - INPUT DATA IS INCONSISTENT'///)
0075              STOP = -2.
0076              GO TO 330
                C
                C              WRITE INPUT
                C
0077            185 WRITE(6,190)
0078            190 FORMAT(//20X,'INPUT', //)
0079              WRITE(6,200) NUMNDS
0080            200 FORMAT(10X,'NUMBER OF NODES = ', I5)
0081              WRITE(6,210) NUMEGS
0082            210 FORMAT(10X,'NUMBER OF EDGES =',I5)
0083              WRITE(6,220) M
0084            220 FORMAT(10X,'NUMBER OF DISTRICTS = ',I5)
0085              WRITE(6,230) IDEPOT
0086            230 FORMAT(10X,'DEPOT IS NODE',I5)
0087              WRITE(6,240) C
0088            240 FORMAT(10X,'TRUCK CAPACITY =', F10.3)
0089              WRITE(6,250) STOP
0090            250 FORMAT(10X,'TERMINATION COST =',F10.3)
0091              WRITE(6,255) EPSILN
0092            255 FORMAT(10X,'EPSILON =', F10.3)
0093              WRITE (6,260)
0094            260 FORMAT(//20X,'NETWORK EDGES'//)
0095              WRITE(6,270)
0096            270 FORMAT( 7X,'EDGE',6X,'IEND',6X,'JEND',8X,'COST',11X, 'LOAD'/)
0097              DO 290 KK = 1,NUMEGS
0098            290 WRITE(6,280) KK, IEND(KK),JEND(KK), COST(KK),ELOAD(KK)
0099            280 FORMAT(5X,I5,2I10,2F14.3)
0100              WRITE(6,300)
0101            300 FORMAT(//20X,'AROUND MATRIX' //)
0102              WRITE (6,305)
0103            305 FORMAT(7X,'NODE',10X,'INCIDENT NODES'/)
0104              DO 310 I = 1,NUMNDS
0105              NUM = NUMINC(I)
0106            310 WRITE(6,320) I, (AROUND(I,J), J = 1,NUM)
0107            320 FORMAT(5X,I5,5X,13I5)
0108            330 RETURN
0109              END
```

```
0001              SUBROUTINE SEC2
                C
                C              SECTION 2  CREATE A UNICURSAL GRAPH BY ADDING
                C              RETRACED EDGES FOUND BY THE KMATCH SUBROUTINE
                C
0002              COMMON DIST(90,90),IPATH(90,90),JPATH(90),IEND(270),
                1JEND(270),COST(270),NODD(90),ELOAD(270),NSOURC,NTERM,INUM,
                2 NUMNDS,NUMEGS,INODE(1080),JJNODE(1080),AROUND(90,10),NUMINC(90),
                3 INNODE(60,10),CYCNDS(60),SP(60,60),NRSTND(60,60),CLOAD(60),
                4 DLOAD(20),IN(20,40),NODE(90),ND(20),JX(60,60),ITIE(100),NF(100),
                5 SPB(60),M,N,TCOST,INF,FINF,NCYNDS,IDEPOT,AVLOAD,C,NATCH,
```

```
     6 NEXTND(10),ITOUR(20,300),NOTOUR(20) ,STOP,IMOUT,COSTK(20)
0003  COMMON/KMOTPT/IKM(1080),JKM(1080),NONDS,NOEDGS, MATCH(1090),
     1 EDGWT(1080)
0004  COMMON/ STOREE / STORE(90,90)
0005  INTEGER AROUND,CYCNDS
C
C          INITIALIZATION NECESSARY FOR SUBROUTINE PATHS
C
0006       DO 401 I = 1,NUMNDS
0007       DO 400 K = 1,NUMNDS
0008       DIST(I,K) = FINF
0009   400 IPATH(I,K) = K
0010   401 DIST(I,I) = 0.
0011       DO 420 KK = 1,NUMEGS
0012       DIST(JEND(KK),IEND(KK)) = COST(KK)
0013   420 DIST(IEND(KK),JEND(KK)) = COST(KK)
0014       DO 423 I = 1,NUMNDS
0015       DO 423 J = 1,NUMNDS
0016   423 STORE(I,J) = DIST(I,J)
0017       CALL PATHS
C
C          FIND ALL NODES OF ODD DEGREE
C
0018       NONDS = 0
0019       J = 0
0020       DO 440 I = 1,NUMNDS
0021       KOUNT  = 0
0022       DO 430 KK = 1,NUMEGS
0023       IF(JEND(KK) .EQ. I) GO TO 425
0024       IF(IEND(KK) .NE. I) GO TO 430
0025   425 KOUNT = KOUNT + 1
0026   430 CONTINUE
0027       IF(2*(KOUNT/2) .EQ. KOUNT) GO TO 440
0028       NONDS = NONDS + 1
0029       NODD(NONDS) = I
0030   440 CONTINUE
0031       IF(NONDS .NE. 0) GO TO 445
C
C          BYPASS KMATCH IF NO ODD NODES EXIST
C
0032       MATCH = 0
0033       RETURN
C
C          NOW CREATE EDGES TO BE READ INTO KMATCH
C
0034   445 KK = 0
0035       N1 = NONDS -1
0036       DO 450 J = 1,N1
0037       N2 = NONDS-J
0038       DO 450 I = 1,N2
0039       KK = KK + 1
0040       IINODE(KK) = NODD(J)
0041       JJNODE(KK) = NODD(J+ I)
0042       IKM(KK) = J
0043       JKM(KK) = J + I
0044   450 EDGWT(KK) =-DIST(NODD(J),NODD(J + I)) + 10000.
0045       NOEDGS = KK
0046       CALL KMATCH
0047       MATCH = 0
0048       DO 470 KK = 1,NOEDGS
C
C          IS EDGE KK IN MATCHING
C
0049       IF(MATCH (KK) .NE. 1) GO TO 470
0050       NSOURC = IINODE(KK)
0051       NTERM = JJNODE(KK)
0052       CALL DECODE
C
C          ADD EDGES TO GRAPH G1 TO GET G3 I.E. EVEN UP THE ORIGINAL
C          GRAPH
C
0053       N1 = INUM - 1
0054       DO 460 I = 1,N1
0055       NUMEGS = NUMEGS + 1
0056       IEND(NUMEGS) = JPATH(I)
0057       JEND(NUMEGS) = JPATH(I + 1)
C
C          IF AN EDGE IS RETRACED, ADD TO THE AROUND MATRIX
C
```

```
0058            IENDN= IEND(NUMEGS)
0059            JENDN= JEND(NUMEGS)
0060            NUMI = NUMINC(IENDN)
0061            DO 500 JJ = 1,NUMI
0062            IF(AROUND(IENDN,JJ) .EQ. JENDN) GO TO 505
0063          500 CONTINUE
0064          505 NUMINC(IENDN) = NUMINC(IENDN) + 1
0065            NUMI = NUMINC(IENDN)
0066            JJ2 = JJ + 2
0067            DO 510 III = JJ2,NUMI
0068            II = NUMI + JJ2 -III
0069          510 AROUND(IENDN,II) = AROUND(IENDN,II - 1)
0070            AROUND(IENDN,JJ + 1) = JENDN
0071            NUMJ = NUMINC(JENDN)
0072            DO 520 JJ = 1,NUMJ
0073            IF(AROUND(JENDN,JJ) .EQ.IENDN) GO TO 525
0074          520 CONTINUE
0075          525 NUMINC(JENDN) = NUMINC(JENDN) + 1
0076            NUMJ = NUMINC(JENDN)
0077            JJ2 = JJ + 2
0078            DO 530 III = JJ2,NUMJ
0079            II = NUMJ + JJ2 -III
0080          530 AROUND(JENDN,II) = AROUND(JENDN,II - 1)
0081            AROUND(JENDN,JJ + 1) = IENDN
                C
                C               END OF ALTERATION OF AROUND MATRIX
                C
                C               DETERMINE IF EDGE(1) IS MATCHED
                C
0082            IF (NATCH .EQ. 1) GO TO 458
0083            IF(IEND(NUMEGS) .EQ.IEND(1)) GOTO 452
0084            IF(IEND(NUMEGS) .NE. JEND(1)) GOTO 458
0085            IF(JEND(NUMEGS) .NE. IEND(1)) GO TO 458
0086            GO TO 456
0087          452 IF(JEND(NUMEGS) .NE. JEND(1))  GO TO 458
0088          456 NATCH = 1
0089          458 COST(NUMEGS) = DIST(JPATH(I),JPATH(I + 1))
0090          460 ELOAD(NUMEGS)  = 0.
0091          470 CONTINUE
                C
                C               INTERMEDIATE OUTPUT ROUTINE
                C
0092            IF(IMOUT .NE . 1) GO TO 850
0093            WRITE (6,810)
0094          810 FORMAT('1',20X,  'INTERMEDIATE OUTPUT'//)
0095            WRITE(6,820)
0096          820 FORMAT( 7X,'EDGE',6X,'IEND',6X,'JEND',8X,'COST',11X, 'LOAD'/)
0097            DO 480 KK = 1,NUMEGS
0098          480 WRITE(6,800) KK, IEND(KK),JEND(KK), COST(KK),ELOAD(KK)
0099          800 FORMAT(5X,I5,2I10,2F14.3)
0100          850 RETURN
0101            END
```

```
0001            SUBROUTINE SEC3
                C
                C               SECTION 3       MODIFY GRAPH TO ONE OF CYCLE NODES
                C
0002            COMMON DIST(90,90),IPATH(90,90),JPATH(90),IEND(270),
                1JEND(270),COST(270),NODD(90),ELOAD(270),NSOURC,NTERM,INUM,
                2 NUMNDS,NUMEGS,IINODE(1080),JJNODE(1080),AROUND(90,10),NUMINC(90),
                3 INNODE(60,10),CYCNDS(60),SP(60,60),NRSTND(60,60),CLOAD(60),
                4 DLOAD(20),IN(20,40),NODE(90),NO(20),JX(60,60),ITIE(100),NF(100),
                5 SPB(60),M,N,TCOST,INF,FINF,NCYNDS,IDEPOT,AVLOAD,C,NATCH,
                6 NEXTND(10),ITOUR(20,300),NOTOUR(20) ,STOP,IMOUT,COSTK(20)
0003            COMMON/KMOTPT/IKM(1080),JKM(1080),NONDS,NOEDGS, MATCH(1080),
                1 EDGWT(1080)
0004            INTEGER AROUND,CYCNDS
0005            CALL LABEL
0006            IF(IMOUT .NE. 1) GO TO 2303
0007            WRITE(6,2302)
0008          2302 FORMAT(//10X,'AROUND(I,J)'//)
0009            DO 2301 I = 1,NUMNDS
0010          2301 WRITE(6,2030)(AROUND(I,J),J = 1,10)
0011          2030 FORMAT(10I5)
0012          2303 DO 2305 I = 1,NUMNDS
0013            DO 2300 J = 1,10
```

```
0014          2300 INNODE(I,J) = 0
0015          2305 CYCNDS(I) = 0
                  C
                  C               PUT IEND(1) AND JEND(1) IN THE FIRST CYCLE NODE
                  C
0016                   I1 = IEND(1)
0017                   J1 = JEND(1)
0018                   INNODE(2,1) = I1
0019                   INNODE(2,2) = J1
0020                   AROUND(I1,1 ) = INF
                  C
                  C               INITIALIZE FOR THE REST OF CYCLE NODE 1 WHEN EDGE
                  C               (IEND(1),JEND(1)) ISNT RETRACED
                  C
0021                   IF(NATCH .EQ. 1) GOTO 2105
0022                   AROUND(I1,NUMINC(I1)) = INF
0023                   KOUNT = 1
0024                   K = 2
0025                   KK = 2
0026                   II = 1
0027                   JK = 1
0028                   GO TO 2165
0029          2105 INNODE(2,3) = I1
                  C
                  C               EDGE (IEND(1),JEND(1)) IS MATCHED AND THEREFORE IS THE
                  C               FIRST CYCLE NODE ITSELF
                  C
0030                   AROUND(I1,2) = INF
0031                   NUJ = NUMINC(J1)
0032                   DO 2100 JJ = 1,NUJ
0033                   IF(AROUND(J1,JJ) .EQ. I1) GO TO 2107
0034          2100 CONTINUE
0035          2107 AROUND(J1,JJ) = INF
0036                   IF(JJ .EQ. 1) GO TO 2108
0037                   AROUND(J1,JJ - 1) = INF
0038                   GO TO 2109
0039          2108 AROUND(J1,NUMINC(J1)) = INF
0040          2109 CYCNDS(2) = 2
0041                   K = 3
0042                   KK = 2
0043                   KOUNT  =2
0044          2110 II = 0
0045          2111 II = II + 1
                  C
                  C               FIND A NEGATIVE EDGE IN AROUND MATRIX TO BRANCH ON
                  C               (TO START ANEW CYCLE NODE)
                  C
0046          2112 JK = 0
0047          2113 JK = JK + 1
0048                   IF(AROUND(INNODE(KK,II),JK) .GE. 0) GO TO 2120
                  C
                  C               DETERMINE FIRST 2 NODES IN CYCLENODES
                  C
0049                   INNODE(K,1) = INNODE(KK,II)
0050                   INNK1 = INNODE(K,1)
0051                   INNODE(K,2) = -AROUND(INNK1,JK)
0052                   AROUND(INNK1,JK) = INF
                  C
                  C               INDICATE WHICH EDGES AROUND A NODE HAVE BEEN PUT IN A
                  C               CYCLE NODE
                  C
0053                   IF(NATCH .NE. 1) GOT O 2130
0054                   IF(JK .EQ. NUMINC(INNK1)) GO TO 2140
0055                   AROUND(INNK1,JK + 1) = INF
0056                   GO TO 2160
0057          2140 AROUND(INNK1,1) = INF
0058                   GO TO 2160
0059          2130 IF(JK .EQ. 1) GO TO 2150
0060                   AROUND(INNK1,JK-1) = INF
0061                   GO TO 2160
0062          2150 AROUND(INNK1,NUMINC(INNK1)) = INF
0063          2160 KOUNT = KOUNT + 1
0064          2165 DO 2220 J = 2,50
                  C
                  C               FIND THE ELEMENT IN AROUND ROW WHICH CORRESPONDS TO THE
                  C               ROW WE CAME FROM
                  C
0065                   INNKJ = INNODE(K,J)
0066                   NUI = NUMINC(INNKJ)
0067                   DO 2170 JJJ = 1,NUI
0068                   JJ = NUI  + 1 - JJJ
0069                   IF(AROUND(INNKJ,JJ) .EQ. INNODE(K,J-1)) GO TO 2185
0070          2170 CONTINUE
```

```
0071          2185 IF(NATCH .NE. 1) GO TO 2180
0072               IF(JJ .NE. 1) GO TO 2190
0073               INNODE(K,J + 1) = -AROUND(INNKJ,NUMINC(INNKJ))
0074               AROUND(INNKJ,NUMINC(INNKJ)) = INF
0075               GO TO 2210
0076          2190 INNODE(K,J + 1) = -AROUND(INNKJ, JJ - 1)
0077               AROUND(INNKJ,JJ - 1) = INF
0078               GO TO 2210
                C
                C              SET NEXT NODE IN CYCLE NODE
                C
0079           2180 IF(JJ .EQ. NUMINC(INNKJ)) GO TO 2200
0080               INNODE(K,J + 1) = -AROUND(INNKJ,JJ + 1)
0081               AROUND(INNKJ,JJ + 1) = INF
0082               GO TO 2210
0083           2200 INNODE(K,J + 1) = -AROUND(INNKJ,1)
0084               AROUND(INNKJ,1) = INF
0085           2210 KOUNT = KOUNT + 1
0086               AROUND(INNKJ,JJ) = INF
                C
                C              IS THE LOOP CLOSED
                C
0087               IF(INNODE(K,J + 1) .EQ. INNODE(K,1)) GO TO 2230
0088           2220 CONTINUE
0089           2230 CYCNDS(K) = J
                C
                C              ARE ALL THE EDGES IN A CYCLE NODE
                C
0090               IF(KOUNT .EQ. NUMEGS) GO TO 2240
0091               K =K + 1
0092           2120 IF(JK .NE. NUMINC(INNODE(KK,II)))GO TO 2113
0093               IF(II .NE. CYCNDS(KK )) GO TO 2111
0094               KK = KK + 1
0095               GO TO 2110
0096           2240 NCYNDS = K
                C
                C              CALCULATE THE EDGE LENGHTS BETWEEN THE CYCLE NODES JUST
                C              CREATED BY FINDING THE SHORTEST, SHORTEST PATH BETWEEN
                C              ALL PAIRS OF NODES IN THE CYCLE NODES
                C
0097               DO 2350 I = 1,NCYNDS
0098               DO 2350 J = 1,NCYNDS
0099               NRSTND (I,J) = 0
0100           2350 SP(I,J) = FINF
0101               INNODE(1,1) =IDEPOT
0102               CYCNDS(1) = 1
0103               N1 = NCYNDS - 1
0104               DO 2370 I = 1,N1
0105               KI = I + 1
0106               DO 2370 K = KI,NCYNDS
0107               TEMPSP = FINF
0108               ICYC = CYCNDS(I)
0109               DO 2360 J = 1,ICYC
0110               KCYC = CYCNDS(K)
0111               DO 2360 L = 1,KCYC
0112               IF(DIST(INNODE(I,J),INNODE(K,L)) .GE. TEMPSP) GO TO 2360
0113               TEMPSP = DIST(INNODE(I,J),INNODE(K,L))
0114               JSAVE = J
0115               LSAVE = L
0116           2360 CONTINUE
0117               SP(I,K) = TEMPSP
0118               SP(K,I) = TEMPSP
0119               NRSTND(I,K) = INNODE(K,LSAVE)
0120               NRSTND(K,I) = INNODE(I,JSAVE)
0121           2370 CONTINUE
                C
                C              CALCULATE THE CYCLE NODE LOADS AND THE AVERAGE LADD
                C              PER TRUCK
                C
0122               CLOAD(1) = 0.
0123               AVLOAD = 0.
0124               NCY1 = NCYNDS + 1
0125               DO 2375 K = 2,NCY1
0126               CLOAD(K) = 0.
0127               KCYC = CYCNDS(K)
0128               DO 2375 I = 1,KCYC
0129               DO 2372 J = 1, NUMEGS
0130               IF(IEND(J) .EQ. INNODE(K    , I)) GO TO 2371
0131               IF(JEND(J) .NE. INNODE(K    ,I)) GO TO     2372
0132               IF(IEND(J) .EQ. INNODE(K    ,I + 1)) GO TO 2373
0133               GO TO 2372
```

```
0134          2371 IF(JEND(J) .NE. INNODE(K   ,I + 1)) GO TO 2372
0135          2373 IF ( ELOAD(J) - 0. .GT. .0001) GO TO 2374
0136               INNODE(K,I + 1) = -INNODE(K,I +1)
0137          2374 CLOAD(K) = ELOAD(J) + CLOAD(K)
0138               ELOAD(J) = 0.
0139               GO TO2375
0140          2372 CONTINUE
0141          2375 CONTINUE
0142               DO 2376 K = 2,NCY1
0143          2376  AVLOAD = AVLOAD + CLOAD(K)
0144               AVLOAD = AVLOAD/M
                C
                C              MAKE SURE THE FIRST INNODE IS NEG IF  THE LAST + 1 IS
                C
0145               DO 100 K = 2,NCYNDS
0146               IF( INNODE(K,CYCNDS(K) + 1) .GT. 0) GO TO 100
0147               INNODE(K,1) = -INNODE(K,1)
0148            100 CONTINUE
                C
                C              INTERMEDIATE OUTPUT ROUTINE
                C
0149               IF(IMOUT .NE. 1) GO TO 2395
0150               WRITE(6,2383)
0151          2383 FORMAT(//10X,'NUMINC(J)'//)
0152               WRITE(6,2382)(NUMINC(J),J = 1,NUMNDS)
0153          2382 FORMAT(10X,14I5)
0154               WRITE(6,2381)
0155          2381 FORMAT(//10X,'SP(I,J)'//)
0156               DO 2380 I = 1,NCYNDS
0157          2380 WRITE(6,2390)(SP(I,J),J = 1,NCYNDS)
0158          2390 FORMAT(1X,10F7.3)
0159               WRITE(6,2391)
0160          2391 FORMAT(//10X,'INNODE(I,J)'//)
0161               DO 2250 I = 1,NCYNDS
0162          2250 WRITE(6,2010)(INNODE(I,J),J = 1,10)
0163          2010 FORMAT(1X,10I5)
0164               WRITE(6,2255)
0165          2255 FORMAT(//10X,'CYCNDS(I)'//)
0166               WRITE(6,2010)(CYCNDS(J),J = 1,K)
0167               WRITE(6,2052)
0168          2052 FORMAT(//10X,'NRSTND(I,J)'//)
0169               DO 2051 I = 1,NCYNDS
0170          2051 WRITE(6,2010)(NRSTND(I,J),J = 1,NCYNDS)
0171               WRITE(6,2060)
0172          2060 FORMAT(//10X,'CLOAD(J)'//)
0173               WRITE(6,2061)(CLOAD(J),J = 1,NCYNDS)
0174          2061 FORMAT(1X,10F6.2)
0175          2395 RETURN
0176               END
```

```
0001               SUBROUTINE SEC4
                C
                C              SECTION 4   SCREEN AND BUILD THE CAPACITATED
                C              SPANNING TREE
                C
0002               COMMON DIST(90,90),IPATH(90,90),JPATH(90),IEND(270),
                  1JEND(270),COST(270),NODD(90),ELOAD(270),NSOURC,NTERM,INUM,
                  2 NUMNDS,NUMEGS,IINODE(1080),JJNODE(1080),AROUND(90,10),NUMINC(90),
                  3 INNODE(60,10),CYCNDS(60),SP(60,60),NRSTND(60,60),CLOAD(60),
                  4 DLOAD(20),IN(20,40),NODE(90),NO(20),JX(60,60),ITIE(100),NF(100),
                  5 SPB(60),M,N,TCOST,INF,FINF,NCYNDS,IDEPOT,AVLOAD,C,NATCH,
                  6 NEXTND(10),ITOUR(20,300),NOTOUR(20) ,STOP,IMOUT,COSTK(20)
0003               COMMON/KMOTPT/IKM(1080),JKM(1080),NONDS,NOEDGS, MATCH(1090),
                  1 EDGWT(1080)
0004               COMMON/BILD/INTREE (50,20),IRETRN,NUMBER,IRMS ,HBEGIN,IBEGIN,
                  1 IONE,ALIST(200),NEXT(200),IPACK(200,2), MASK(30),INJX(60),IDONE
0005               INTEGER HBEGIN
0006            100 DO 165 I = 1,NCYNDS
0007               NODE(I) = 0
0008               DO 165 J = 1,NCYNDS
0009            165 JX(I,J) = 0
                C
                C              REARRANGE LABELS ON CYCLE  NODES SUCH THAT
                C              SP(1,NODE(J)) .LE. SP(1,NODE(J + 1))
                C
0010               DO 1010 I = 2,NCYNDS
```

```
0011          1010 SPB(I) = SP(1,I)
0012               DO 1030 J = 2,NCYNDS
0013               T3 = FINF-9.
0014               DO 1020 I = 2,NCYNDS
0015               IF(SPB(I) .GE. T3) GO TO 1020
0016               T3 = SPB(I)
0017               NODE(J) = I
0018          1020 CONTINUE
0019          1030 SPB(NODE(J)) = FINF
0020               NODE(1) = 1
0021               CALL ARMS
0022               IF (STOP .LT. 0.) GO TO 240
0023               DO 140 K = 1,M
0024           140 DLOAD(K) = CLOAD(IN(K,1))
0025               JTEMP = NUMNDS
0026               NUMNDS = NCYNDS
0027               DO 200 J = 1,NCYNDS
0028               DO 200 I = 1,NCYNDS
0029               IF(SP(I,J) - 0. .GT. .00001) GO TO 202
0030               DIST(I,J) = CLOAD(I)
0031               GO TO 200
0032           202 DIST(I,J) = FINF
0033           200 CONTINUE
0034               DO 220 J = 1,NCYNDS
0035           220 DIST(1,J) = FINF
                C
                C              ELIMINATE EDGES LEADING FROM ARM NODES
                C
0036               DO 230 L = 1,M
0037               DO 230 I = 1,NCYNDS
0038           230 DIST(IN(L,1),I) = FINF
0039               JPATH(1) = 0
                C
                C              INITIALIZE INTREE
                C
0040               DO 235 I = 1,NCYNDS
0041               DO 235 J = 1,M
0042           235 INTREE(I,J) = 0
0043               CALL SCREEN
0044               IF(IRETRN .EQ . 1) GO TO 100
0045               CALL BUILD
0046               IF(IRETRN .EQ . 1) GO TO 100
0047               NUMNDS = JTEMP
0048           240 RETURN
0049               END
```

```
0001               SUBROUTINE SEC5
                C
                C              SECTION 5  EXCHANGE   CYCLE NODES FROM SMALL DISTRICTS
                C              TO LARGE ONES IN ORDER TO BALANCE THE LOADS
                C
0002               COMMON DIST(90,90),IPATH(90,90),JPATH(90),IEND(270),
                  1JEND(270),COST(270),NODD(90),ELOAD(270),NSOURC,NTERM,INUM,
                  2 NUMNDS,NUMEGS,IINODE(1080),JJNODE(1080),AROUND(90,10),NUMINC(90),
                  3 INNODE(60,10),CYCNDS(60),SP(60,60),NRSTND(60,60),CLOAD(60),
                  4 DLOAD(20),IN(20,40),NODE(90),NO(20),JX(60,60),ITIE(100),NF(100),
                  5 SPB(60),M,N,TCOST,INF,FINF,NCYNDS,IDEPOT,AVLOAD,C,NATCH,
                  6 NEXTND(10),ITOUR(20,300),NOTOUR(20) ,STOP,IMOUT,COSTK(20)
0003               COMMON/KMOTPT/IKM(1080),JKM(1080),NONDS,NOEDGS, MATCH(1080),
                  1 EDGWT(1080)
0004               COMMON/BILD/INTREE (50,20),IRETRN,NUMBER,IRNS ,HBEGIN,IBEGIN,
                  1 IONE,ALIST(200),NEXT(200),IPACK(200,2), MASK(30),INJX(60),IDONE
0005               COMMON /EPSIL/ EPSILN ,IFLAG
0006               ICRIT2 = 0
0007               IRETRN =0
0008               DO 90 I = 1,NCYNDS
0009            90 INJX(I) = 1
0010           100 IDID = 0
                C
                C              KOUNT COUNTS THE NUMBER OF TIMES WE RUN THE SUBROUTINE
                C              AND LATER IF KOUNT IS GT IDONE TERMINATES W/OUT A FEASIBLE
                C              TREE
                C
0011               N = NCYNDS
0012               CALL ORDERK
0013               IF(DLOAD(M) - C .LT. .00001) GOTO 300
0014               IRETRN = 1
```

```
0015              DO 185 K = 1,M
0016              DO 185 L1 = 1,M
0017              L = M + 1- L1
0018              IF(K .EQ.L) GO TO 185
0019              NOL = NO(L)
0020              IF (NOL .LT. 2) GO TO 185
0021              NOK = NO(K)
0022              DO 180 I = 1,NOK
0023              DO 180 J =  2,NOL
                  C
                  C               CHECK IF EDGES EXIST
                  C
0024              IF(SP(IN(K,I), IN(L,J)) - 0. .GT. .00001) GO TO 180
                  C
                  C               IS BALANCE IMPROVABLE  (CRITERION 2)
                  C
0025              IF (IFLAG .NE. 1) GO TO 1110
0026              IF( ICRIT2 .EQ. 0) GO TO1110
0027              IF ( DLOAD(K) .GT. AVLOAD) GO TO 1110
0028              IF(DLOAD(L) .GT. AVLOAD) GO TO 185
0029              GO TO 1120
0030             1110 IF(DLOAD(L) .LT. AVLOAD) GO TO 185
0031             1120 IF(ABS(DLOAD(K) + 2*CLOAD(IN(L,J)) - DLOAD(L)) .LT. ABS(DLOAD(K)
                     1  -DLOAD(L)))GO TO 1130
0032              GO TO 180
                  C
                  C               WILL BALANCE BE IMPROVED (CRITERION 1)
                  C
0033             1100 IF(ABS(DLOAD(K) + CLOAD(IN(L,J)) -AVLOAD) + ABS(DLOAD(L) -
                     1 CLOAD(IN(L,J)) - AVLOAD) -  ( ABS(DLOAD(K) - AVLOAD)
                     2 + ABS(DLOAD(L) - AVLOAD)) .GT.-.001  ) GO TO 180
                  C
                  C               INITIALIZE DIST
                  C
0034             1130 DO 120 II = 1,NCYNDS
0035              DO 120 JJ = 1,NCYNDS
0036             120 DIST(II,JJ) = FINF
                  C
                  C               SET UP FOR SHORTEST PATH ROUTINE AND CALL PATHS
                  C
0037              DO 140 II = 1,NOL
0038              IF(II .EQ. J) GO TO 140
0039              DO 130 JJ = 1,NOL
0040              IF(JJ .EQ. J) GO TO 130
0041              IF(SP(IN(L,II), IN(L,JJ))- 0. .GT. .00001) GO TO 130
0042              DIST(II,JJ) = 0.
0043             130 CONTINUE
0044             140 CONTINUE
0045              JTEMP = NUMNDS
0046              NUMNDS = NOL
0047              CALL PATHS
0048              NUMNDS = JTEMP
                  C
                  C               CHECK IF LARGEST DISTRICT (L) WITH NODE REMOVED
                  C               IS CONNECTED
                  C
0049              DO 170 JJJ = 2,NOL
0050              IF (DIST(  1,JJJ ) - 0. .LT. .00001) GO TO 170
0051              IF (JJJ .NE. J) GO TO 180
0052             170 CONTINUE
0053              GO TO 190
0054             180 CONTINUE
0055             185 CONTINUE
0056              IF(IDID .EQ. 1) GO TO 100
0057              IF (IFLAG .NE. 1) GO TO 175
0058              IF(ICRIT2) 1010,1010,175
0059             1010 ICRIT2 = 1
0060              GO TO 100
0061             175 IF(INOUT .NE. 1) GO TO 300
0062              DO 186 K = 1,M
0063              IF(DLOAD(K) .GT. C + EPSILN) GO TO 187
0064             186 CONTINUE
0065              WRITE(6,188)
0066             188 FORMAT(//10X,'TEMPORARY SOLUTION -  LOADS DONT EXCEED C + EPSILON'
                    1 , //)
0067              CALL SEC6
0068              CALL SEC7
0069              DO 189 I = 1,NCYNDS
0070              DO 189 J = 1,NCYNDS
0071             189 NRSTND(I,J) = IABS(NRSTND(I,J))
0072             187 GO TO 300
                 C
```

```
C             ADD A NODE TO SMALLER DISTRICT
C
0073        190 NO(K) = NO(K) + 1
0074            IN(K,NO(K)) = IN(L,J)
0075            JX(IN(K,NO(K)),IN(K,I) ) = K
0076            JX(IN(K,I), IN(K,NO(K)))= K
0077            DLOAD(K) = DLOAD(K) + CLOAD(IN(K,NO(K)))
0078            DLOAD(L) = DLOAD(L) - CLOAD(IN(L,J))
0079            IDID = 1
C
C                 SUBTRACT NODE FROM LARGE DISTRICT
C
0080            N1 = NO(L) -1
0081            DO 200 JI = J,N1
0082        200 IN(L,JI) = IN(L,JI + 1)
0083            DO 205 JI = 1,N1
0084        205 INJX(IN(L,JI)) = 0
0085            DO 210 MM = 1,NCYNDS
0086            DO 210 NN = 1,NCYNDS
0087            IF(JX(MM,NN) .NE. L) GO TO 210
0088            JX(MM,NN) = 0
0089        210 CONTINUE
C
C                 REBUILD SUBTREE FOR DISTRICT L
C
0090            JX(1,IN(L,1)) = L
0091            JX(IN(L,1),1) = L
0092            INJX(IN(L,1)) = 1
0093            IN(L,NO(L)) = 0
0094            NO(L) = NO(L) -1
0095            NOL = NO(L)
0096            NUMBER = 1
0097        220 DO 270 II = 1,NOL
0098            IF(INJX (IN(L,II)) .NE. 1) GO TO 270
0099            DO 260 JJ = 1,NOL
0100            IF(INJX(IN(L,JJ)) .EQ. 1) GO TO 260
0101            IF(SP(IN(L,II),IN(L,JJ)) -D. .GT. .00001) GO TO 260
0102            NUMBER = NUMBER + 1
0103            JX(IN(L,II) ,IN(L,JJ)) = L
0104            JX (IN(L,JJ),IN(L,II)) = L
0105            INJX(IN(L,JJ)) = 1
0106            IF(NUMBER .EQ. NOL) GO TO 280
0107        260 CONTINUE
0108        270 CONTINUE
0109            IF(NUMBER .NE. NOL) GO TO 220
C
C                 CHECK IF DISTRICT LOADS ARE FEASIBLE
C
0110        280 DO 290 KK = 1,M
0111            K1 = M + 1 - KK
0112            IF(DLOAD(K1) - C .GT. .00001) GO  TO 100
0113        290 CONTINUE
0114            IRETRN = 0
0115        300 RETURN
0116            END
```

```
0001            SUBROUTINE SEC6
C
C                 SECTION 6    DECODE THE CAPACITATED SPANNING
C                 TREE INTO TOURS
C
0002            COMMON DIST(90,90),IPATH(90,90),JPATH(90),IEND(270),
               1JEND(270),COST(270),NODD(90),ELOAD(270),NSOURC,NTERM,INUM,
               2 NUMNDS,NUMEGS,IINODE(1080),JJNODE(1080),AROUND(90,10),NUMINC(90),
               3 INNODE(60,10),CYCNDS(60),SP(60,60),NRSTND(60,60),CLOAD(60),
               4 DLOAD(20),IN(20,40),NODE(90),ND(20),JX(60,60),ITIE(100),NF(100),
               5 SPB(60),M,N,TCOST,INF,FINF,NCYNDS,IDEPOT,AVLOAD,C,NATCH,
               6 NEXTND(10),ITOUR(20,300),NOTOUR(20) ,STOP,IMOUT,COSTK(20)
0003            COMMON/KMOTPT/IKM(1080),JKM(1080),NONDS,NOEDGS, MATCH(1090),
               1 EDGWT(1080)
0004            COMMON/ STOREE / STORE(90,90)
0005            INTEGER CYCNDS
0006            DO 90 I = 1,NUMNDS
0007            DO 90 J = 1,NUMNDS
0008        90  DIST(I,J) = STORE(I,J)
0009            CALL PATHS
```

```
0010              DO 100 J = 1,200
0011              DO 100 I = 1,M
0012            100 ITOUR(I,J) = 0
0013              DO 110 I = 1,10
0014            110 NEXTND(I) = 0
0015              DO 210 K = 1,M
0016              ITOUR(K,1) = IDEPOT
0017              NOWND  = NODE(1)
0018              NOK = NO(K)
0019              DO 130 J = 1,NOK
0020              IF(JX(NOWND,IN(K,J)) .EQ. K) GO TO 140
0021            130 CONTINUE
                C
                C               SET A NODE IN THE NEXT NODE AS THE SECOND NODE IN THE
                C               TOUR
                C
0022            140 ITOUR(K,2) =-NRSTND(NOWND,IN(K,J))
0023              NRSTND(NOWND,IN(K,J)) = -NRSTND(NOWND,IN(K,J))
0024              JTEMP = IN(K,J)
0025              JX(NOWND,IN(K,J)) = -JX(NOWND,IN(K,J))
0026              JX(IN(K,J),NOWND) = -JX(IN(K,J),NOWND)
0027              II = 2
                C
                C               RELABLE PRESENT CYCLE NODES
                C
0028            150 LASTND = NOWND
0029              NOWND = JTEMP
                C
                C               WHAT NUMBER IS NODE IN SEQUENCE IN CYCLE NODE
                C
0030              NCYCNN = CYCNDS(NOWND)
0031              DO 160 JJ = 1,NCYCNN
0032              IF(ITOUR(K,II) .EQ. -IABS(INNODE(NOWND,JJ))) GO TO 170
0033            160 CONTINUE
                C
                C               FIND CYCLE NODES ADJACENT TO NOWND IN TREE
                C
0034            170 I = 0
0035              DO 180 J = 1,NOK
0036              IF(JX(NOWND,IN(K,J)) .NE. K) GO TO 180
0037              I =I+ 1
0038              NEXTND(I) = IN(K,J)
0039            180 CONTINUE
0040              NUMNEX = I
                C
                C               CONTINUE THE TOUR AROUND THE SEQUENCE OF NODES IN THE
                C               CYCLE NODE, BRANCHING TO A NOW CYCLE NODE IF THE NRSTND
                C               OF THAT CYCLE EQUALS THE NODE WE ARE AT
                C
0041              NCYCJJ= JJ + CYCNDS(NOWND)
0042              DO 190 J = JJ,NCYCJJ
0043              IF(J .GT. NCYCNN ) GO TO 200
0044              L = J
0045              GO TO 210
0046            200 L = J -NCYCNN
0047            210 IF(NUMNEX .EQ. 0) GO TO 250
0048              DO 230 I = 1,NUMNEX
0049              IF (NRSTND (NEXTND(I),NOWND) .EQ. IABS(INNODE(NOWND,L))) GO TO 300
0050            230 CONTINUE
0051              II = II +1
0052              ITOUR(K,II) = INNODE(NOWND,L  + 1)
0053              GO TO 240
0054            250 II = II + 1
0055              ITOUR(K,II)  = INNODE(NOWND,  L + 1)
0056              IF(IABS(ITOUR(K,II)).EQ. ITOUR(K,1)) GO TO 310
                C
                C               GO BACK TO A PREVOUUS CYCLE NODE IE. THE TREE HAS BEEN
                C               TOURED BEYOND THIS CYCLE  NODE
                C
0057            240 DO 260 LL = 1,NOK
0058              IF(NRSTND(IN(K,LL),NOWND) .EQ.-IABS(ITOUR(K,II)))GO TO 290
0059            260 CONTINUE
0060              IF(NRSTND(1,NOWND).EQ . -IABS(ITOUR(K,II)))GO TO 280
0061            190 CONTINUE
0062            290 II = II + 1
0063              ITOUR(K,II) =-NRSTND(NOWND,IN(K,LL))
0064              JTEMP = IN(K,LL)
0065              IF(ITOUR(K,II) .EQ.-ITOUR(K,1)) GO TO 310
0066              GO TO 150
                C
                C               BRANCH TO THE NEAREST  NODE IN THE NEXT CYCLE NODE
                C
```

```
0067          300 II = II + 1
0068              ITOUR(K,II) =-NRSTND(NOWND,NEXTND(I))
0069              NRSTND(NOWND,NEXTND(I)) = -NRSTND(NOWND,NEXTND(I))
0070              JX(NEXTND(I),NOWND) = -JX(NEXTND(I),NOWND)
0071              JX(NOWND,NEXTND(I)) = -JX(NOWND,NEXTND(I))
0072              JTEMP = NEXTND(I)
0073              IF(ITOUR(K,II) .NE.-ITOUR(K,1)) GO TO 150
0074          280 II = II + 1
0075              ITOUR(K,II) =-IDEPOT
0076          310 NOTOUR (K) = II
                C
                C             ELIMINATE DUPLICATION IN ITOUR LIST
                C
0077              DO 360 K = 1,M
0078              NOT1 = NOTOUR(K) -1
0079              DO 360 II = 1,NOT1
0080              JJ = 0
0081          345 JJ = JJ + 1
0082              IF(ITOUR(K,II) .EQ. IABS(ITOUR(K,II + JJ))) GO TO 345
0083              IF(JJ .EQ. 1 ) GO TO 360
0084              III  =II + 1
0085              DO 350 J = III,NOT1
0086          350 ITOUR(K,J) = ITOUR(K,J + JJ - 1)
0087              NOTOUR(K) = NOTOUR(K) - JJ + 1
0088          360 NOT1 = NOTOUR(K) -1
                C
                C             DECODE THE ITOUR LIST SO THE NODES IN THE TRAVELED
                C             SHORTEST PATHS ARE ALSO LISTED (A NEGATIVE NODE MEANS
                C             THE EDGE  ENTERENG THAT NODE WAS TRAVELED NOT SERVICED)
                C
0089              DO 400 K = 1,M
0090              II = 0
0091          510 II = II + 1
0092              IF(ITOUR(K,II) .GT. 0) GO TO 500
0093              NOTII = NOTOUR(K) - II
0094              DO 520 I = 1,NOTII
0095              IF(ITOUR(K,II + I) .GT. 0) GO TO 530
0096          520 CONTINUE
0097              I = I + 1
0098          530 IF(I-1 .EQ. 0)GO TO 500
0099              NOTOUR(K) = NOTOUR(K) - I + 1
0100              NOT = NOTOUR(K)
0101              DO 550 J = II,NOT
0102          550 ITOUR(K,J) = ITOUR(K,J + I - 1)
0103          500 IF(II .LT. NOTOUR(K) - 1) GO TO 510
0104              II = 0
0105          410 II = II + 1
0106              IF(ITOUR(K,II) .GE. 0) GO TO 440
0107              NSOURC = ITOUR (K,II - 1)
0108              NTERM = -ITOUR(K,II)
0109              CALL DECODE
0110              IF(INUM .LE. 2) GO TO 440
0111              NOTOUR(K) = NOTOUR(K) + INUM - 2
0112              IIIN = II + INUM - 2
0113              NOTK = NOTOUR(K)
0114              DO 420 JJ1 = IIIN,NOTK
0115              JJ = NOTK + IIIN - JJ1
0116          420 ITOUR(K,JJ) = ITOUR(K,JJ - (INUM-2))
0117              KK = 0
0118              IN1 = INUM - 1
0119              DO 430 JJ = 2,IN1
0120              KK = KK + 1
0121          430 ITOUR(K,II - 1 + KK) = -JPATH(JJ)
0122              II = II + INUM - 2
0123          440 IF(II .NE. NOTOUR(K)) GO TO 410
0124          400 CONTINUE
0125              IF(IOUT .NE. 1) GO TO 600
0126              WRITE (6,610)
0127          610 FORMAT(//10X,'ITOUR(K,J)',//)
0128              DO 80 I = 1,M
0129              NOT  = NOTOUR(I)
0130           80 WRITE(6,97)(ITOUR(I,J),J = 1,NOT)
0131           97 FORMAT(2X,26I3)
0132          600 RETURN
0133              END
```

```
0001            SUBROUTINE SEC7
                C
                C
                C           SECTION 7  OUTPUT ROUTINE
                C
0002            COMMON DIST(90,90),IPATH(90,90),JPATH(90),IEND(270),
               1JEND(270),COST(270),NODD(90),ELOAD(270),NSOURC,NTERM,INUM,
               2 NUMNDS,NUMEGS,IINODE(1080),JJNODE(1080),AROUND(90,10),NUMINC(90),
               3 INNODE(60,10),CYCNDS(60),SP(60,60),NRSTND(60,60),CLOAD(60),
               4 DLOAD(20),IN(20,40),NODE(90),NO(20),JX(60,60),ITIE(100),NF(100),
               5 SPB(60),M,N,TCOST,INF,FINF,NCYNDS,IDEPOT,AVLOAD,C,NATCH,
               6 NEXTND(10),ITOUR(20,300),NOTOUR(20) ,STOP,IMOUT,COSTK(20)
0003            INTEGER  HBEGIN, AROUND, CYCNDS
0004            WRITE(6,100)
0005          100  FORMAT ('1',19X,'OUTPUT'//)
0006            DO 105 K = 1,M
0007          105 COSTK(K) = 0.
0008            DO 400 K = 1,M
0009            NOT = NOTOUR(K)
0010            WRITE (6,110) K
0011          110 FORMAT(//5X,'TOUR',I3/)
0012            I = 2
0013            IF(ITOUR(K,I) .LT. 0) GO TO 160
0014          140 ISTART = I
0015            IBEG = ITOUR(K,I - 1)
0016          145 I = I + 1
0017            IF(I .EQ. NOT + 1) GO TO 150
0018            IF(ITOUR(K,I).GT.0)GO TO 145
0019            IFINI = I -1
0020            GOT O 155
0021          150 IFINI = NOT
0022          155 WRITE(6,120) IBEG
0023          120 FORMAT(8X,'SERVICE FROM NODE',I3)
0024            WRITE(6,130) (ITOUR(K,J), J = ISTART,IFINI)
0025          130 FORMAT(10X,'TO NODE(S)',2013)
0026            IF(IFINI .EQ. NOT) GO TO 240
0027          160 ISTART = I
0028            IBEG = ITOUR(K,I - 1)
0029          190 I = I+ 1
0030            IF(I .EQ. NOT + 1) GO TO 200
0031            IF(ITOUR(K,I) .LT. 0) GO TO 190
0032            IFINI = I - 1
0033            GO TO 210
0034          200 IFINI = NOT
0035          210 WRITE(6,170) IBEG
0036          170 FORMAT(8X,'TRAVEL FROM NODE', I3)
0037            DO 175 J = ISTART,IFINI
0038          175 ITOUR (K,J) = -ITOUR(K,J)
0039            WRITE(6 ,180) (ITOUR(K,J) ,J = ISTART,IFINI)
0040          180 FORMAT(10X,'TO NODE(S)' ,2013)
0041            COSTK(K) = COSTK(K) + DIST(IBEG, ITOUR(K,IFINI))
0042            IF(IFINI .EQ. NOT) GO TO 240
0043            GO TO 140
0044          240 WRITE(6,250) COSTK(K)
0045          250 FORMAT(/8X,'TRAVELED DISTANCE =', F10.2)
0046            WRITE(6,260) DLOAD(K)
0047          260 FORMAT(8X,'LOAD =',F10.2)
0048          400 CONTINUE
0049            RETURN
0050            END
```

```
0001            SUBROUTINE PATHS
                C
                C           PATHS   FINDS THE SHORTEST PATHS BETWEEN ALL PAIRS
                C           OF NODES
                C
0002            COMMON DIST(90,90),IPATH(90,90),JPATH(90),IEND(270),
               1JEND(270),COST(270),NODD(90),ELOAD(270),NSOURC,NTERM,INUM,
               2 NODES ,NUMEGS,IINODE(1080),JJNODE(1080),AROUND(90,10),NUMINC(90),
               3 INNODE(60,10),CYCNDS(60),SP(60,60),NRSTND(60,60),CLOAD(60),
               4 DLOAD(20),IN(20,40),NODE(90),NO(20),JX(60,60),ITIE(100),NF(100),
               5 SPB(60),M,N,TCOST,INF,FINF,NCYNDS,IDEPOT,AVLOAD,C,NATCH,
               6 NEXTND(10),ITOUR(20,300),NOTOUR(20) ,STOP,IMOUT,COSTK(20)
0003            DO 40 I = 1,NODES
0004            DO 40 K = 1,NODES
0005          40 IPATH(I,K) = K
```

```
0006            DO 50 J = 1,NODES
0007            DO 50 I = 1,NODES
0008            DO 50 K = 1,NODES
0009            DTEMP = DIST(I,J) + DIST(J,K)
0010            IF(DIST(I,K) .LE. DTEMP) GO TO 50
0011            DIST(I,K) = DTEMP
0012            IPATH(I,K) = IPATH(I,J)
0013         50 CONTINUE
0014            RETURN
0015            END
```

```
0001            SUBROUTINE DECODE
                C
                C          DECODE DETERMINES THE SEQUENCE OF EDGES IN THE
                C          PATHS  FOUND IN SUBROUTINE PATHS
                C
0002            COMMON DIST(90,90),IPATH(90,90),JPATH(90),IEND(270),
                1JEND(270),COST(270),NODD(90),ELOAD(270),NSOURC,NTERM,INUM,
                2 NUMNDS,NUMEGS,IINODE(1080),JJNODE(1080),AROUND(90,10),NUMINC(90),
                3 INNODE(60,10),CYCNDS(60),SP(60,60),NRSTND(60,60),CLOAD(60),
                4 DLOAD(20),IN(20,40),NODE(90),NO(20),JX(60,60),ITIE(100),NF(100),
                5 SPB(60),M,N,TCOST,INF,FINF,NCYNDS,IDEPOT,AVLOAD,C,NATCH,
                6 NEXTND(10),ITOUR(20,300),NOTOUR(20) ,STOP,IMOUT,COSTK(20)
0003            INUM = 1
0004            JPATH(INUM) = NSOURC
0005            I = NSOURC
0006         90 INUM = INUM + 1
0007            I = IPATH(I,NTERM)
0008            JPATH(INUM) = I
0009            IF(I .NE.NTERM) GOTO 90
0010            RETURN
0011            END
```

```
0001            SUBROUTINE KMATCH
                C SUBROUTINE IS A MODIFICATION OF KMATCH                                    KMCH
                C ALGORITHM AND CODING BY LEE J. WHITE, OHIO STATE UNIVERSITY               KMCH
                C MODIFIED TO ASA FORTRAN IV BY JON LIEBMAN, UNIVERSITY OF ILLINOIS         KMCH
                C
                C          KMATCH AND OUTPUT FIND EDGES TO BE RETRACED IN ORDER
                C          TO CREATE A UNICURSAL GRAPH
                C
0002            REAL            NDWT(70),LRGWT,Z(70)
0003            INTEGER                 NDLBL(70),MATCH(1080),TRND(46),TREDG(46),
                1 CURDEL,TMPDEL,EDGE,WRKND,ZS,ROOT(70),HIHEST,OUTBLS(46),
                2 INBLS(70),BLSND(70),BLSEDG(70),DISTND(46),
                3 HIBLS,SAVND,CHANGE(46),STACKA(70),
                4 STACKB(70),PTRA,PTRB,SWITCH,ACTND,BLSAT,
                5 BLSLBL(70),PARORD(70),ONODE(70),BLOBLS,
                6 FAREND,WRKEDG,COUNT,WORKM,SWITA,WORKQ,
                7 SWITB,SWITC,OUTMCH                                                        KMCH
0004            EQUIVALENCE(Z,NDWT),(NDLBL,BLSLBL),(INBLS,PARORD),                          KMCH
                1 (ROOT,ONODE),(KEY,SWITC)                                                  KMCH
0005            COMMON DIST(90,90),IPATH(90,90),JPATH(90),NZ(270),                          KMCH
                1  NZ(270),COST(270),NODD(90),ELOAD(270),NSOURC,NTERM,INUM,
                2 NUMNDS,NUMEGS,II    (1080),JJ    (1080),AROUND(90,10),NUMINC(90),
                3 INNODE(60,10),CYCNDS(60),SP(60,60),NRSTND(60,60),CLOAD(60),
                4 DLOAD(20),ZZ(20,40),NODE(90),NO(20),JX(60,60),ITIE(100),NF(100),
                5 SPB(60),M,N,TCOST,INF,FINF,NCYNDS,IDEPOT,AVLOAD,C,NATCH,
                6 NEXTND(10),ITOUR(20,300),NOTOUR(20) ,STOP,IMOUT,COSTK(20)
0006            COMMON/OTPT/ SCALAR, NDWT,MATCH,
                1 TOTEWT,K,HIHEST,TRND,TREDG,NDLBL,                                         KMCH
                2 OUTBLS,INBLS,ROOT,BLSND,BLSEDG,DISTND,LOBLS,HIBLS,                        KMCH
                3      IN,IOUT                                                              KMCH
0007            COMMON/KMOTPT/  I(1080),  J(1080),NONDS,NOEDGS,OUTMCH(1080),
                1 EDGWT(1080)
0008            IN=5
0009            IOUT=6                                                                      KMCH
0010            LOBLS=NONDS+1                                                               KMCH
0011            HIBLS=(NONDS-1)/2+NONDS                                              036    KMCH
                C                                                                    037    KMCH
                C SET UP HIHEST AND TOTEWT FOR OUTPUT SUBROUTINE                     038    KMCH
                                                                                     039    KMCH
```

```
                C
0012                  IF(HIBLS.GE.NOEDGS)GO TO 251
0013                  HIHEST=NOEDGS
0014                  GO TO 253
0015              251 HIHEST=HIBLS
0016              253 TOTEWT=0.
0017                  DO 254 KK=1,NOEDGS
0018              254 TOTEWT=TOTEWT+EDGWT(KK)
                C
                C FIND LARGEST EDGE WEIGHT
                C
0019                  LRGWT=0.
0020                  DO 7 KK=1,NOEDGS
0021                  IF(EDGWT(KK).LE.LRGWT)GO TO 7
0022                  LRGWT=EDGWT(KK)
0023                7 CONTINUE
0024                  K=0
0025                  NUMX=0
                C
                C SET UP NDWT AND TREES
                C
0026              252 DO 8 KK=1,NONDS
0027                  TRND(KK)=0
0028                  TREDG(KK)=0
0029                  DISTND(KK)=0
0030                  OUTBLS(KK)=0
0031                  INBLS(KK)=0
0032                  CHANGE(KK)=0
0033                  NDWT(KK)=LRGWT/2.
0034                8 CONTINUE
                C SET UP MATCH
                C SET UP MATCH
                C NDLBL,AND SCALAR
                C
0035                  DO 9 KK=1,NOEDGS
0036                  MATCH(KK)=0
0037                9 CONTINUE
0038                  DO 18 KK=1,NONDS
0039                  NDLBL(KK)=1
0040                  ROOT(KK)=KK
0041               18 CONTINUE
0042                  DO 1020 KK=LOBLS,HIBLS
0043                  Z(KK)=0.
0044                  PARORD(KK)=-1
0045                  BLSLBL(KK)=0
0046            1020 ONODE(KK)=0
0047                  DO 1021 KK=1,HIBLS
0048                  BLSND(KK)=0
0049            1021 BLSEDG(KK)=0
0050                  SCALAR=LRGWT/2.
0051               88 CALL OUTPUT
0052            8421 CURMIN=LRGWT+1.0
0053                  CURDEL=6
0054                  DO 1025 KK=LOBLS,HIBLS
0055                  IF(PARORD(KK).NE.0)GO TO 1025
0056                  IF(BLSLBL(KK).NE.3)GO TO 1025
0057                  TMPMIN=Z(KK)/2.
0058                  TMPDEL=1
0059                  IF(TMPMIN-CURMIN.GT.-.00005)GO TO 1025
0060                  CURMIN=TMPMIN
0061                  CURDEL=TMPDEL
0062                  BLOBLS=KK
0063            1025 CONTINUE
0064                  DO 25 KK=1,NOEDGS
0065                  IK=I(KK)
0066                  JK=J(KK)
0067                  IF(NDLBL(IK).EQ.4.AND.NDLBL(JK).LE.2)GO TO 33
0068                  IF(NDLBL(IK).GE.3)GO TO 25
0069                  IF(NDLBL(JK).GE.3) GO TO 30
0070                  IF(MATCH(KK).EQ.1)GO TO 25
0071                  TMPMIN=(-EDGWT(KK)+NDWT(IK   )+NDWT(JK   ))/2.
0072                  IF(ROOT(IK).EQ.ROOT(JK))GO TO 1027
0073                  TMPDEL=2
0074                  GO TO 40
0075               30 IF(NDLBL(JK).EQ.3) GO TO 25
0076               33 IF(MATCH(KK).EQ.1)GO TO 25
0077                  TMPMIN=-EDGWT(KK)+NDWT(IK)+NDWT(JK)
0078                  TMPDEL=3
0079                  GO TO 40
0080            1027 IF(OUTBLS(IK).NE.OUTBLS(JK))GO TO 1040
0081                  IF(OUTBLS(IK).NE.0) GO TO 25
0082            1040 TMPDEL=4
```

```
040 KMCH
041 KMCH
042 KMCH
043 KMCH
044 KMCH
045 KMCH
046 KMCH
047 KMCH
048 KMCH
049 KMCH
050 KMCH
051 KMCH
052 KMCH
053 KMCH
054 KMCH
055 KMCH
056 KMCH
057 KMCH
058 KMCH
059 KMCH
060 KMCH
061 KMCH
062 KMCH
063 KMCH
064 KMCH
065 KMCH
066 KMCH
067 KMCH
068 KMCH
069 KMCH
070 KMCH
070 KMCH
071 KMCH
072 KMCH
073 KMCH
074 KMCH
075 KMCH
076 KMCH
077 KMCH
078 KMCH
079 KMCH
080 KMCH
081 KMCH
082 KMCH
083 KMCH
084 KMCH
085 KMCH
086 KMCH
087 KMCH
088 KMCH
089 KMCH
090 KMCH
091 KMCH
092 KMCH
093 KMCH
094 KMCH
095 KMCH
096 KMCH
097 KMCH
098 KMCH
099 KMCH
100 KMCH
101 KMCH
102 KMCH
103 KMCH
104 KMCH
105 KMCH
106 KMCH
107 KMCH
108 KMCH
109 KMCH
110 KMCH
111 KMCH
112 KMCH
113 KMCH
114 KMCH
115 KMCH
116 KMCH
117 KMCH
118 KMCH
119 KMCH
120 KMCH
```

```
                C                                                                  121 KMCH
                C  IS TH NEW DELTA AN IMPROVEMENT                                  122 KMCH
                C                                                                  123 KMCH
0083               40 IF(THPMIN-CURMIN.GT..00005)GO TO 25                          124 KMCH
0084                  IF(THPMIN-CURMIN.LT.-.00005)GO TO 42                         125 KMCH
0085                  IF(THPDEL.GE.CURDEL)GO TO 25                                 126 KMCH
0086               42 CURMIN=THPMIN                                                127 KMCH
0087                  CURDEL=THPDEL                                                128 KMCH
0088                  EDGE=KK                                                      129 KMCH
0089               25 CONTINUE                                                     130 KMCH
                C                                                                  131 KMCH
                C  STOP CONDITION - NO VALID DELTA                                 132 KMCH
                C                                                                  133 KMCH
0090                  IF(CURMIN.EQ.LRGWT+1.)GO TO 610                              134 KMCH
                C                                                                  135 KMCH
                C  ADJUST NDWT                                                     136 KMCH
                C                                                                  137 KMCH
0091                  DO 48 KK=1,NONDS                                             138 KMCH
0092                  IF(NDLBL(KK).GE.3)GO TO 49                                   139 KMCH
0093                  NDWT(KK)=NDWT(KK)-CURMIN                                     140 KMCH
0094                  GO TO 48                                                     141 KMCH
0095               49 IF(NDLBL(KK).EQ.4)GO TO 48                                   142 KMCH
0096                  NDWT(KK)=NDWT(KK)+CURMIN                                     143 KMCH
0097               48 CONTINUE                                                     144 KMCH
                C                                                                  145 KMCH
                C  ADJUST Z                                                        146 KMCH
                C                                                                  147 KMCH
0098                  DO 1030 KK=LOBLS,HIBLS                                       148 KMCH
0099                  IF(PARORD(KK).NE.0)GO TO 1030                                149 KMCH
0100                  IF(BLSLBL(KK).EQ.3)GO TO 1031                                150 KMCH
0101                  IF(BLSLBL(KK).EQ.4)GO TO 1030                                151 KMCH
0102                  Z(KK)=Z(KK)+2.*CURMIN                                        152 KMCH
0103                  GO TO 1030                                                   153 KMCH
0104            1031  Z(KK)=Z(KK)-2.*CURMIN                                        154 KMCH
0105            1030  CONTINUE                                                     155 KMCH
                C                                                                  156 KMCH
                C  ADJUST SCALAR                                                   157 KMCH
                C                                                                  158 KMCH
0106                  SCALAR=SCALAR-CURMIN                                         159 KMCH
                C                                                                  160 KMCH
                C  NOW STARTING WORKING WITH TREES                                 161 KMCH
                C                                                                  162 KMCH
0107                  GO TO(2100,970,1052,1300),CURDEL                             163 KMCH
                C                                                                  164 KMCH
                C  DELTA 2       AUGMENT                                           165 KMCH
                C                                                                  166 KMCH
0108               970 K=K+1                                                       167 KMCH
0109                  NUMX=1                                                       168 KMCH
0110                  Z5=0                                                         169 KMCH
0111                  WRKND=I(EDGE)                                                170 KMCH
0112            2603  DO 1125 KK=1,NONDS                                           171 KMCH
0113            1125  CHANGE(KK)=0                                                 172 KMCH
0114                  IF(CURDEL.EQ.2)GO TO 79                                      173 KMCH
0115                  IF(CURDEL.EQ.4)GO TO 79                                      174 KMCH
0116                  MATCH(EDGE)=0                                                175 KMCH
0117               109 SAVND=WRKND                                                 176 KMCH
0118                  GO TO 1105                                                   177 KMCH
0119                79 MATCH(EDGE)=1                                               178 KMCH
                C                                                                  179 KMCH
                C  STARTING HERE WE TREAT A VERTEX FOLLOWED BY A DOTTED EDGE.      180 KMCH
                C  IF AT THIS POINT WE HAVE A BLOSSOM IT CAN'T BE A ROOT BECAUSE   181 KMCH
                C  ONLY DOTTED EDGES CAN LEAD INTO A BLOSSOM WHICH IS A ROOT.      182 KMCH
                C                                                                  183 KMCH
0120               70 IF(DISTND(WRKND).NE.0)GO TO 1100                             184 KMCH
0121                  IF(TRND(WRKND).EQ.0)GO TO 71                                 185 KMCH
0122            1100  IDW=DISTND(WRKND)                                           186 KMCH
0123                  ITW=TREDG(WRKND)                                             187 KMCH
0124                  IF(IDW.NE.0)GO TO 1103                                       188 KMCH
0125                  MATCH(ITW)=0                                                 189 KMCH
0126                  SAVND=TRND(WRKND)                                            190 KMCH
0127                  WRKND=TRND(WRKND)                                            191 KMCH
0128                  GO TO 1105                                                   192 KMCH
                C  IF BLOSSOM                                                      193 KMCH
0129            1103  IF(TREDG(IDW).NE.0)GO TO 11031                               194 KMCH
0130                  CHANGE(IDW)=WRKND                                            195 KMCH
0131                  GO TO 7114                                                   196 KMCH
0132           11031  ITT=TREDG(IDW)                                               197 KMCH
0133                  MATCH(ITT)=0                                                 198 KMCH
0134                  IIT=I(ITT)                                                   199 KMCH
0135                  IF(OUTBLS(IIT).EQ.OUTBLS(WRKND))GO TO 1107                   200 KMCH
0136                  SAVND=IIT                                                    201 KMCH
```

```
0137              CHANGE(IDW)=WRKND                                          202 KMCH
0138              GO TO 1108                                                 203 KMCH
0139             1107 SAVND=J(ITT)                                           204 KMCH
0140              CHANGE(IDW)=WRKND                                          205 KMCH
0141             1108 WRKND=TRND(IDW)                                        206 KMCH
                 C STARTING HERE WE TREAT A VERTEX FOLLOWED BY A SOLID EDGE  207 KMCH
0142             1105 IF(DISTND(WRKND).NE.0)GO TO 1111                       208 KMCH
0143              ITW=TREDG(WRKND)                                           209 KMCH
0144              MATCH(ITW)=1                                               210 KMCH
0145              WRKND=TRND(WRKND)                                          211 KMCH
0146              GO TO 70                                                   212 KMCH
0147             1111 IDW=DISTND(WRKND)                                      213 KMCH
0148              CHANGE(IDW)=SAVND                                          214 KMCH
                 C BRANCH IF BLOSSOM IS THE ROOT                            215 KMCH
0149              IF(TRND(IDW).EQ.0)GO TO 71                                 216 KMCH
0150              ITW=TREDG(IDW)                                             217 KMCH
0151              MATCH(ITW)=1                                               218 KMCH
0152              WRKND=TRND(IDW)                                            219 KMCH
0153              GO TO 70                                                   220 KMCH
                 C                                                          221 KMCH
                 C AT THE ROOT                                              222 KMCH
0154                71 IF(DISTND(WRKND).NE.0)GO TO 1113                     223 KMCH
0155              GO TO 7114                                                 224 KMCH
                 C THE NEXT 2 STATEMENTS ARE FOR BLOSSOM AUGMENTS           225 KMCH
0156             7114 IF(Z5.EQ.1)GO TO 1113                                  226 KMCH
0157              IF(CURDEL.GE.4)GO TO 2020                                  227 KMCH
                 C DESTROY THE TREE ,RELEABEL THE NODES                     228 KMCH
0158             1113 DO 3020 KK=LOBLS,HIBLS                                 229 KMCH
0159              IF(PARORD(KK).NE.0)GO TO 3020                              230 KMCH
0160              IDW=DNODE(KK)                                              231 KMCH
0161              IF(ROOT(IDW).NE.ROOT(WRKND))GO TO 3020                     232 KMCH
0162              BLSLBL(KK)=4                                               233 KMCH
0163             3020 CONTINUE                                               234 KMCH
0164              IROOT=ROOT(WRKND)                                          235 KMCH
0165              DO 550 KK=1,NONDS                                          236 KMCH
0166              IF(ROOT(KK).NE.IROOT)GO TO 550                             237 KMCH
0167              NDLBL(KK)=4                                                238 KMCH
0168              ROOT(KK)=0                                                 239 KMCH
0169              TRND(KK)=0                                                 240 KMCH
0170              TREDG(KK)=0                                                241 KMCH
0171              IF(DISTND(KK).EQ.0)GO TO 550                               242 KMCH
0172              IDW=DISTND(KK)                                             243 KMCH
0173              IF(CHANGE(IDW).EQ.0)GO TO 550                              244 KMCH
0174              DISTND(KK)=CHANGE(IDW)                                     245 KMCH
0175               550 CONTINUE                                              246 KMCH
                 C UPDATE DNODE FOR OUTERMOST BLOSSOMS INVOLVED IN          247 KMCH
                 C THAT TREE.                                               248 KMCH
0176              DO 1120 KK=LOBLS,HIBLS                                     249 KMCH
0177              IF(PARORD(KK).NE.0)GO TO 1120                              250 KMCH
0178              IDW=DNODE(KK)                                              251 KMCH
0179              IF(CHANGE(IDW).EQ.0)GO TO 1120                             252 KMCH
0180              DNODE(KK)=CHANGE(IDW)                                      253 KMCH
0181             1120 CONTINUE                                               254 KMCH
0182             2020 IF(Z5.EQ.1)GO TO 88                                    255 KMCH
0183              WRKND=J(EDGE)                                              256 KMCH
0184              Z5=1                                                       257 KMCH
0185              GO TO 2603                                                 258 KMCH
                 C                                                          259 KMCH
                 C                                                          260 KMCH
                 C  DELTA 3                                                 261 KMCH
                 C                                                          262 KMCH
                 C                                                          263 KMCH
0186             1052 KEY=3                                                 264 KMCH
0187              Z5=0                                                       265 KMCH
0188              GO TO 971                                                  266 KMCH
0189              972 KEY=2                                                  267 KMCH
0190              971 WORKM=I(EDGE)                                          268 KMCH
0191              IF(NDLBL(WORKM).EQ.4)GO TO 110                             269 KMCH
0192              WORKQ=J(EDGE)                                              270 KMCH
0193              GO TO 111                                                  271 KMCH
0194              110 WORKM=J(EDGE)                                          272 KMCH
0195              WORKQ=I(EDGE)                                              273 KMCH
0196              111 IF(OUTBLS(WORKQ).NE.0)GO TO 1062                       274 KMCH
                 C RECORD WORKM REGARDLESS OF WETHER OR NOT IT IS IN A BLOSSOM 275 KMCH
                 C BLOSSOM                                                  276 KMCH
0197              TRND(WORKQ)=WORKM                                          277 KMCH
0198              TREDG(WORKQ)=EDGE                                          278 KMCH
0199              ROOT(WORKQ)=ROOT(WORKM)                                    279 KMCH
0200              NDLBL(WORKQ)=KEY                                           280 KMCH
0201              GO TO 106                                                  281 KMCH
0202             1062 IDW=DISTND(WORKQ)                                      282 KMCH
```

```
0203              TRNO(IDW)=WORKN                                                  283 KMCH
0204              TREDG(IDW)=EDGE                                                  284 KMCH
0205              DO 1072 KK=1,NONDS                                               285 KMCH
0206              IF(OUTBLS(KK).NE.OUTBLS(WORKQ))GO TO 1072                        286 KMCH
0207              ROOT(KK)=ROOT(WORKM)                                             287 KMCH
0208              NDLBL(KK)=KEY                                                    288 KMCH
0209           1072 CONTINUE                                                       289 KMCH
0210              IDW=OUTBLS(WORKQ)                                                290 KMCH
0211              BLSLBL(IDW)=KEY                                                  291 KMCH
0212            106 IF(I5.EQ.1)GO TO 88                                            292 KMCH
0213              DO 107 KKK=1,NOEDGS                                              293 KMCH
0214              IK=I(KKK)                                                        294 KMCH
0215              JK=J(KKK)                                                        295 KMCH
0216              IF(NDWT(IK)+NDWT(JK).EQ.EDGWT(KKK).AND.MATCH(KKK).EQ.1)GO TO 103 296 KMCH
0217              GO TO 107                                                        297 KMCH
0218            103 IF(I(KKK).EQ.WORKQ)GO TO 7001                                  298 KMCH
0219              IF(I(KKK).EQ.DISTND(WORKQ))GO TO 7001                            299 KMCH
0220              IF(J(KKK).EQ.DISTND(WORKQ))GO TO 7001                            300 KMCH
0221              IF(J(KKK).NE.WORKQ)GO TO 107                                     301 KMCH
0222           7001 I5=1                                                           302 KMCH
0223              EDGE=KKK                                                         303 KMCH
0224              GO TO 972                                                        304 KMCH
0225            107 CONTINUE                                                       305 KMCH
0226              GO TO 88                                                         306 KMCH
                C                                                                  307 KMCH
                C                                                                  308 KMCH
                C FORM BLOSSOM                                                     309 KMCH
                C                                                                  310 KMCH
                C                                                                  311 KMCH
                C TRACE BACK TO ROOT FROM BOTH VERTICES                            312 KMCH
0227            1300 SWITCH=0                                                      313 KMCH
0228              PTRA=1                                                           314 KMCH
0229              PTRB=1                                                           315 KMCH
0230              ACTND=I(EDGE)                                                    316 KMCH
0231            1303 IF(DISTND(ACTND).EQ.0)GO TO 1302                              317 KMCH
0232              ACTND=DISTND(ACTND)                                              318 KMCH
0233            1302 IF(SWITCH.EQ.1)GO TO 1304                                     319 KMCH
                C THE STACKS HOLD NODES NO DISTNDS AS OPPOSED TO ELEMEN            320 KMCH
                C ENTS.                                                            321 KMCH
0234              STACKA(PTRA)=ACTND                                               322 KMCH
0235              IF(TRND(ACTND).EQ.0)GO TO 1305                                   323 KMCH
0236              PTRA=PTRA+1                                                      324 KMCH
0237              ACTND=TRND(ACTND)                                                325 KMCH
0238              GO TO 1303                                                       326 KMCH
0239            1304 STACKB(PTRB)=ACTND                                            327 KMCH
0240              IF(TRND(ACTND).EQ.0)GO TO 1306                                   328 KMCH
0241              PTRB=PTRB+1                                                      329 KMCH
0242              ACTND=TRND(ACTND)                                                330 KMCH
0243              GO TO 1303                                                       331 KMCH
0244            1305 SWITCH=1                                                      332 KMCH
0245              ACTND=J(EDGE)                                                    333 KMCH
0246              GO TO 1303                                                       334 KMCH
                C FIND ROOT OF BLOSSOM                                             335 KMCH
0247            1306 IF(PTRA.EQ.0)GO TO 1310                                       336 KMCH
0248              IF(PTRB.EQ.0)GO TO 1310                                          337 KMCH
0249              IF(STACKA(PTRA).NE.STACKB(PTRB))GO TO 1310                       338 KMCH
0250              PTRA=PTRA-1                                                      339 KMCH
0251              PTRB=PTRB-1                                                      340 KMCH
0252              GO TO 1306                                                       341 KMCH
0253            1310 PTRA=PTRA+1                                                   342 KMCH
0254              PTRB=PTRB+1                                                      343 KMCH
                C PTRA POINTS TO BLOSSOM ROOT IN STACKA - SAME FOR B'S             344 KMCH
0255              BLSRT=STACKA(PTRA)                                               345 KMCH
                C CHECK FOR AUGMENT CASE.                                          346 KMCH
                C A NEW BLOSSOM MUST BE FORMED / FIND AN UNUSED BLOSSOM NUMBER     347 KMCH
0256            1312 DO 1314 KK=LOBLS,HIBLS                                        348 KMCH
0257              IF(PARORD(KK).EQ.-1)GO TO 1315                                   349 KMCH
0258            1314 CONTINUE                                                      350 KMCH
0259            1315 NEWBLS=KK                                                     351 KMCH
0260              DO 1353 KK=1,NONDS                                               352 KMCH
0261            1353 CHANGE(KK)=0                                                  353 KMCH
                C SET UP ENTRIES IN BLSND   BLSEDG                                 354 KMCH
                C  GO DOWN STACKA                                                  355 KMCH
0262              PTRA=1                                                           356 KMCH
0263              IW=STACKA(1)                                                     357 KMCH
0264            1323 WORKQ=STACKA(PTRA)                                            358 KMCH
                C NOTE BLSRT IS THE ROOT OF THE NEW BLOSSOM   IS EITHER            359 KMCH
                C A NODE OR A DISTND                                               360 KMCH
0265              CHANGE(WORKQ)=BLSRT                                              361 KMCH
0266              IF(WORKQ.EQ.BLSRT)GO TO 1325                                     362 KMCH
```

```
0267            IF(DISTND(WORKQ).EQ.0)GO TO 1320                        363 KMCH
              C IF WORKQ IS IN ABLOSSOM ALREADY *                       364 KMCH
0268            WORKQ=OUTBLS(WORKQ)                                     365 KMCH
0269            PARORD(WORKQ)=NEWBLS                                    366 KMCH
0270            GO TO 1371                                              367 KMCH
              C                                                         368 KMCH
              C IF WORKQ IS NOT IN A BLOSSOM                            369 KMCH
0271         1320 INBLS(WORKQ)=NEWBLS                                   370 KMCH
0272         1371 IK=STACKA(PTRA)                                       371 KMCH
0273            JK=STACKA(PTRA+1)                                       372 KMCH
0274            IF(DISTND(JK).NE.0)GO TO 1321                           373 KMCH
              C IF NEXT IS NOT IN BLOSSOM ALREADY*                      374 KMCH
0275            BLSND(WORKQ)=JK                                         375 KMCH
0276            GO TO 1322                                              376 KMCH
              C  IF NEXT IS IN A BLOSSOM ALREADY*                       377 KMCH
0277         1321 BLSND(WORKQ)=OUTBLS(JK)                               378 KMCH
0278         1322 ITW=TREDG(IK)                                         379 KMCH
0279            BLSEDG(WORKQ)=ITW                                       380 KMCH
0280            MATCH(ITW)=0                                            381 KMCH
0281            PTRA=PTRA+1                                             382 KMCH
0282            GO TO 1323                                              383 KMCH
              C GO UP STACK B                                           384 KMCH
0283         1325 WORKQ=STACKB(PTRB)                                    385 KMCH
0284            CHANGE(WORKQ)=BLSRT                                     386 KMCH
0285            IF(DISTND(WORKQ).EQ.0)GO TO 1326                        387 KMCH
              C IF WORKQ IS IN BLOSSOM ALREADY*                         388 KMCH
0286            WORKQ=OUTBLS(WORKQ)                                     389 KMCH
0287            PARORD(WORKQ)=NEWBLS                                    390 KMCH
0288            GO TO 1372                                              391 KMCH
              C IF WORKQ IS NOT IN A BLOSSOM YET*                       392 KMCH
0289         1326 INBLS(WORKQ)=NEWBLS                                   393 KMCH
0290         1372 IF(PTRB.EQ.1)GO TO 1330                               394 KMCH
0291            IK=STACKB(PTRB-1)                                       395 KMCH
0292            ITW=TREDG(IK)                                           396 KMCH
0293            IF(DISTND(IK).NE.0)GO TO 1327                           397 KMCH
0294            BLSND(WORKQ)=STACKB(PTRB-1)                             398 KMCH
0295            GO TO 1328                                              399 KMCH
              C IF NEXT IS IN A BLOSSOM ALREADY                         400 KMCH
0296         1327 BLSND(WORKQ)=OUTBLS(IK)                               401 KMCH
0297         1328 BLSEDG(WORKQ)=ITW                                     402 KMCH
0298            MATCH(ITW)=0                                            403 KMCH
0299            PTRB=PTRB-1                                             404 KMCH
0300            GO TO 1325                                              405 KMCH
0301         1330 IF(DISTND(IW).NE.0)GO TO 1331                         406 KMCH
0302            BLSND(WORKQ)=IW                                         407 KMCH
0303            GO TO 1332                                              408 KMCH
0304         1331 BLSND(WORKQ)=OUTBLS(IW)                               409 KMCH
0305         1332 BLSEDG(WORKQ)=EDGE                                    410 KMCH
0306            MATCH(EDGE)=0                                           411 KMCH
              C UPDATE ONODE                                            412 KMCH
0307            ONODE(NEWBLS)=BLSRT                                     413 KMCH
              C MAKE ONE PASS THROUGH THE TREE.                         414 KMCH
0308         1341 DO 1355 KK=1,NONDS                                    415 KMCH
0309            IF(CHANGE(KK).NE.0)GO TO 1360                           416 KMCH
0310            IF(DISTND(KK).EQ.0)GO TO 1355                           417 KMCH
0311            IK=DISTND(KK)                                           418 KMCH
0312            IF(CHANGE(IK).NE.0)GO TO 1360                           419 KMCH
0313            GO TO 1355                                              420 KMCH
0314         1360 NDLBL(KK)=2                                           421 KMCH
0315            OUTBLS(KK)=NEWBLS                                       422 KMCH
0316            IF(DISTND(KK).EQ.BLSRT)GO TO 1355                       423 KMCH
0317            DISTND(KK)=BLSRT                                        424 KMCH
0318            IF(KK.EQ.BLSRT)GO TO 1355                               425 KMCH
0319            TRND(KK)=0                                              426 KMCH
0320            TREDG(KK)=0                                             427 KMCH
0321         1355 CONTINUE                                              428 KMCH
              C UP DATE PARORD   BLSLBL OF NEWBLS                       429 KMCH
0322            PARORD(NEWBLS)=0                                        430 KMCH
0323            BLSLBL(NEWBLS)=2                                        431 KMCH
0324            GO TO 88                                                432 KMCH
              C                                                         433 KMCH
              C                                                         434 KMCH
              C  BLOW UP BLOSSOM                                        435 KMCH
              C                                                         436 KMCH
              C                                                         437 KMCH
              C FIND NEAR ELEMENT (ATTACHED TO SOLID EDGE)   FAR ELEMENT 438 KMCH
              C (ATTACHED TO DOTTED EDGE) -- CALL THEM NEAREND AND      439 KMCH
              C FAREND                                                  440 KMCH
0325         2100 NEREND=ONODE(BLOBLS)                                  441 KMCH
0326            IDALOB=NEREND                                           442 KMCH
0327            IF(INBLS(NEREND).EQ.OUTBLS(NEREND))GO TO 2102           443 KMCH
```

```
0328              NEREND=INBLS(NEREND)                                          444 KMCH
0329           2101 NEND=PAROD(NEREND)                                          445 KMCH
0330              IF(PAROD(NEND).EQ.0)GO TO 2102                                444 KMCH
0331              NEREND=NEND                                                   447 KMCH
0332              GO TO 2101                                                    448 KMCH
0333           2102 WRKEDG=TREDG(IDBLOB)                                        449 KMCH
0334              IW=I(WRKEDG)                                                  450 KMCH
0335              IF(OUTBLS(IW).EQ.BLOBLS)GO TO 2105                            451 KMCH
0336              FAREND=J(WRKEDG)                                              452 KMCH
0337              GO TO 2106                                                    453 KMCH
0338           2105 FAREND=IW                                                   454 KMCH
0339           2106 IF(INBLS(FAREND).EQ.OUTBLS(FAREND))GO TO 2184               455 KMCH
0340              FAREND=INBLS(FAREND)                                          456 KMCH
0341           2109 NEND=PAROD(FAREND)                                          457 KMCH
0342              IF(PAROD(NEND).EQ.0)GO TO 2184                                458 KMCH
0343              FAREND=NEND                                                   459 KMCH
0344              GO TO 2109                                                    460 KMCH
                C UPDATE OUBLS                                                  461 KMCH
0345           2184 DO 2180 KK=1,NONDS                                          462 KMCH
0346              IF(OUTBLS(KK).NE.BLOBLS)GO TO 2180                            463 KMCH
0347              IF (INBLS(KK).EQ.BLOBLS)GO TO 2982                            464 KMCH
0348              WORKQ=INBLS(KK)                                               465 KMCH
0349           2181 NEND=PAROD(WORKQ)                                           466 KMCH
0350              IF(PAROD(NEND).EQ.0)GO TO 2182                                467 KMCH
0351              WORKQ=NEND                                                    468 KMCH
0352              GO TO 2181                                                    469 KMCH
0353           2182 OUTBLS(KK)=WORKQ                                            470 KMCH
0354              GO TO 2180                                                    471 KMCH
0355           2982 OUTBLS(KK)=0                                                472 KMCH
0356           2180 CONTINUE                                                    473 KMCH
                C NOW MAKE A PASS FOR ONODES                                    474 KMCH
                C SWITB=1 INDICATES THAT WE ARE LOOKING FROM AN ELEMENT         475 KMCH
                C TOWARDS AN EDGE WHICH WILL BE LABELED 'SOLID'IN THE           476 KMCH
                C INDUCTION.                                                    477 KMCH
0357              IF(NEREND.LE.NONDS)GO TO 2560                                 478 KMCH
0358              ONODE(NEREND)=ONODE(BLOBLS)                                   479 KMCH
0359           2560 WORKQ=BLSND(NEREND)                                         480 KMCH
0360              SWITB=1                                                       481 KMCH
0361           2172 IF(WORKQ.EQ.NEREND)GO TO 2590                               482 KMCH
0362              IBW=BLSEDG(WORKQ)                                             483 KMCH
0363              IF(WORKQ.GT.NONDS)GO TO 2174                                  484 KMCH
0364              IF(SWITB.EQ.0)GO TO 2173                                      485 KMCH
0365              IF(I(IBW).EQ.WORKQ)GO TO 2171                                 486 KMCH
0366              SAVND=I(IBW)                                                  487 KMCH
0367              GO TO 2170                                                    488 KMCH
0368           2171 SAVND=J(IBW)                                                489 KMCH
0369           2170 MATCH(IBW)=1                                                490 KMCH
0370              WORKQ=BLSND(WORKQ)                                            491 KMCH
0371              SWITB=0                                                       492 KMCH
0372              GO TO 2172                                                    493 KMCH
0373           2173 SWITB=1                                                     494 KMCH
0374              WORKQ=BLSND(WORKQ)                                            495 KMCH
0375              GO TO 2172                                                    496 KMCH
0376           2174 IF(SWITB.EQ.0)GO TO 2177                                    497 KMCH
0377              IB=I(IBW)                                                     498 KMCH
0378              IF(OUTBLS(IB).EQ.WORKQ)GO TO 2175                             499 KMCH
0379              ONODE(WORKQ)=J(IBW)                                           500 KMCH
0380              SAVND=IB                                                      501 KMCH
0381              GO TO 2176                                                    502 KMCH
0382           2175 ONODE(WORKQ)=IB                                             503 KMCH
0383              SAVND=J(IBW)                                                  504 KMCH
0384           2176 MATCH(IBW)=1                                                505 KMCH
0385              SWITB=0                                                       506 KMCH
0386              WORKQ=BLSND(WORKQ)                                            507 KMCH
0387              GO TO 2172                                                    508 KMCH
0388           2177 ONODE(WORKQ)=SAVND                                          509 KMCH
0389              SWITB=1                                                       510 KMCH
0390              WORKQ=BLSND(WORKQ)                                            511 KMCH
0391              GO TO 2172                                                    512 KMCH
                C NOW TRACE FROM BEREND TO FAREND AND FIND IF IT IS THE         513 KMCH
                C ODD CASE OR THE EVEN CASE                                     514 KMCH
0392           2590 WORKQ=NEREND                                                515 KMCH
0393           2110 IF(BLSND(WORKQ).EQ.FAREND)GO TO 2125                        516 KMCH
0394              WORKQ=BLSND(WORKQ)                                            517 KMCH
0395              IF(BLSND(WORKQ).EQ.FAREND)GO TO 2121                          518 KMCH
0396              WORKQ=BLSND(WORKQ)                                            519 KMCH
0397              GO TO 2110                                                    520 KMCH
                C                                                               521 KMCH
```

```
                C EVEN CASE                                                     521 KMCH
                C  IF IT WAS THE EVEN CASE THEN CHANGE IT TO THE                522 KMCH
                C EVEN CASE                                                     524 KMCH
0398             2121 DO 3130 KK=1,HIBLS                                        525 KMCH
0399                  STACKA(KK)=0                                              526 KMCH
0400             3130 STACKB(KK)=0                                              527 KMCH
0401                  WORKQ=NEREND                                              528 KMCH
0402             3131 IBW=BLSND(WORKQ)                                          529 KMCH
0403                  STACKA(IBW)=WORKQ                                         530 KMCH
0404                  STACKB(IBW)=BLSEDG(WORKQ)                                 531 KMCH
0405                  WORKQ=IBW                                                 532 KMCH
0406                  IF(WORKQ.NE.NEREND)GO TO 3131                             533 KMCH
0407                  WORKQ=NEREND                                              534 KMCH
0408             3132 BLSND(WORKQ)=STACKA(WORKQ)                                535 KMCH
0409                  BLSEDG(WORKQ)=STACKB(WORKQ)                               536 KMCH
0410                  WORKQ=STACKA(WORKQ)                                       537 KMCH
0411                  IF(WORKQ.NE.NEREND)GO TO 3132                             538 KMCH
                C ODD PATH                                                      539 KMCH
0412             2135 SWITA=1                                                   540 KMCH
0413                  WORKM=BLSND(NEREND)                                       541 KMCH
0414             2134 IF(WORKM.EQ.FAREND)GO TO 2160                             542 KMCH
0415                  IF(WORKM.GT.NONDS)GO TO 2131                              543 KMCH
                C  IF THIS ELEMENT IS A NODE*                                   544 KMCH
0416                  NDLBL(WORKM)=4                                            545 KMCH
0417                  ROOT(WORKM)=0                                             547 KMCH
0418                  INBLS(WORKM)=0                                            546 KMCH
0419                  DISTND(WORKM)=0                                           548 KMCH
0420                  WORKM=BLSND(WORKM)                                        549 KMCH
0421                  GO TO 2134                                                550 KMCH
                C  IF THIS ELEMENT IS A BLOSSOM*                                551 KMCH
0422             2131 DO 2190 KK=1,NONDS                                        552 KMCH
0423                  IF(OUTBLS(KK).NE.WORKM)GO TO 2190                         553 KMCH
0424                  NDLBL(KK)=4                                               554 KMCH
0425                  ROOT(KK)=0                                                555 KMCH
0426                  DISTND(KK)=DNODE(WORKM)                                   556 KMCH
0427             2190 CONTINUE                                                  557 KMCH
0428                  PARORD(WORKM)=0                                           558 KMCH
0429                  BLSLBL(WORKM)=4                                           559 KMCH
0430                  WORKM=BLSND(WORKM)                                        560 KMCH
0431                  GO TO 2134                                                561 KMCH
                C EVEN PATH                                                     562 KMCH
                C  FIRST A ROUTINE FOR THE FAREND VERTEX                        563 KMCH
0432             2160 IDW=DNODE(WORKM)                                          564 KMCH
0433                  IDB=DNODE(BLOBLS)                                         565 KMCH
0434                  IF(WORKM.GT.NONDS)GO TO 2200                              566 KMCH
0435                  TRND(WORKM)=TRND(IDB)                                     567 KMCH
0436                  TREDG(WORKM)=TREDG(IDB)                                   568 KMCH
0437                  GO TO 2201                                                569 KMCH
0438             2200 TRND(IDW)=TRND(IDB)                                       570 KMCH
0439                  TREDG(IDW)=TREDG(IDB)                                     571 KMCH
0440             2201 SWITC=3                                                   572 KMCH
0441             2205 IF(WORKM.GT.NONDS)GO TO 2210                              573 KMCH
                C  IF THIS IS A NODE*                                           574 KMCH
0442                  NDLBL(WORKM)=SWITC                                        575 KMCH
0443                  INBLS(WORKM)=0                                            576 KMCH
0444                  DISTND(WORKM)=0                                           577 KMCH
0445                  IF(WORKM.EQ.NEREND)GO TO 2450                             578 KMCH
0446                  IBW=BLSND(WORKM)                                          579 KMCH
0447                  IF(IBW.GT.NONDS)GO TO 2203                                580 KMCH
                C  IF NEXT IS A NODE*                                           581 KMCH
0448                  TRND(IBW)=WORKM                                           582 KMCH
0449                  TREDG(IBW)=BLSEDG(WORKM)                                  583 KMCH
0450                  GO TO 2204                                                584 KMCH
                C  IF NEXT IS A BLOSSOM                                         585 KMCH
0451             2203 ID=DNODE(IBW)                                             586 KMCH
0452                  TRND(ID)=WORKM                                            587 KMCH
0453                  TREDG(ID)=BLSEDG(WORKM)                                   588 KMCH
0454             2204 WORKM=BLSND(WORKM)                                        589 KMCH
0455                  IF(SWITC.EQ.2)GO TO 2206                                  590 KMCH
0456                  SWITC=2                                                   591 KMCH
0457                  GO TO 2205                                                592 KMCH
0458             2206 SWITC=3                                                   593 KMCH
0459                  GO TO 2205                                                594 KMCH
                C  IF THIS ELEMENT IS A BLOSSOM                                 595 KMCH
0460             2210 DO 2211 KK=1,NONDS                                        596 KMCH
0461                  IF(OUTBLS(KK).NE.WORKM)GO TO 2211                         597 KMCH
0462                  NDLBL(KK)=SWITC                                           598 KMCH
0463                  DISTND(KK)=DNODE(WORKM)                                   599 KMCH
0464             2211 CONTINUE                                                  600 KMCH
0465                  PARORD(WORKM)=0                                           601 KMCH
0466                  BLSLBL(WORKM)=SWITC                                       602 KMCH
```

```
                C EVEN CASE                                                     522 KMCH
                C IF IT WAS THE EVEN CASE THEN CHANGE IT TO THE                 523 KMCH
                C EVEN CASE                                                     524 KMCH
0398             2121 DO 3130 KK=1,HIBLS                                        525 KMCH
0399                  STACKA(KK)=0                                              526 KMCH
0400             3130 STACKB(KK)=0                                              527 KMCH
0401                  WORKQ=NEREND                                              528 KMCH
0402             3131 IBW=BLSND(WORKQ)                                          529 KMCH
0403                  STACKA(IBW)=WORKQ                                         530 KMCH
0404                  STACKB(IBW)=BLSEDG(WORKQ)                                 531 KMCH
0405                  WORKQ=IBW                                                 532 KMCH
0406                  IF(WORKQ.NE.NEREND)GO TO 3131                             533 KMCH
0407                  WORKQ=NEREND                                              534 KMCH
0408             3132 BLSND(WORKQ)=STACKA(WORKQ)                                535 KMCH
0409                  BLSEDG(WORKQ)=STACKB(WORKQ)                               536 KMCH
0410                  WORKQ=STACKA(WORKQ)                                       537 KMCH
0411                  IF(WORKQ.NE.NEREND)GO TO 3132                             538 KMCH
                C ODD PATH                                                      539 KMCH
0412             2125 SWITA=1                                                   540 KMCH
0413                  WORKM=BLSND(NEREND)                                       541 KMCH
0414             2134 IF(WORKM.EQ.FAREND)GO TO 2160                             542 KMCH
0415                  IF(WORKM.GT.NONDS)GO TO 2131                              543 KMCH
                C IF THIS ELEMENT IS A NODE*                                    544 KMCH
0416                  NDLBL(WORKM)=4                                            545 KMCH
0417                  ROOT(WORKM)=0                                             547 KMCH
0418                  INBLS(WORKM)=0                                            546 KMCH
0419                  DISTND(WORKM)=0                                           548 KMCH
0420                  WORKM=BLSND(WORKM)                                        549 KMCH
0421                  GO TO 2134                                                550 KMCH
                C IF THIS ELEMENT IS A BLOSSOM*                                 551 KMCH
0422             2131 DO 2190 KK=1,NONDS                                        552 KMCH
0423                  IF(OUTBLS(KK).NE.WORKM)GO TO 2190                         553 KMCH
0424                  NDLBL(KK)=4                                               554 KMCH
0425                  ROOT(KK)=0                                                555 KMCH
0426                  DISTND(KK)=DNODE(WORKM)                                   556 KMCH
0427             2190 CONTINUE                                                  557 KMCH
0428                  PARORD(WORKM)=0                                           558 KMCH
0429                  BLSLBL(WORKM)=4                                           559 KMCH
0430                  WORKM=BLSND(WORKM)                                        560 KMCH
0431                  GO TO 2134                                                561 KMCH
                C EVEN PATH                                                     562 KMCH
                C FIRST A ROUTINE FOR THE FAREND VERTEX                         563 KMCH
0432             2160 IDW=DNODE(WORKM)                                          564 KMCH
0433                  IDB=DNODE(BLOBLS)                                         565 KMCH
0434                  IF(WORKM.GT.NONDS)GO TO 2200                              566 KMCH
0435                  TRND(WORKM)=TRND(IDB)                                     567 KMCH
0436                  TREDG(WORKM)=TREDG(IDB)                                   568 KMCH
0437                  GO TO 2201                                                569 KMCH
0438             2200 TRND(IDW)=TRND(IDB)                                       570 KMCH
0439                  TREDG(IDW)=TREDG(IDB)                                     571 KMCH
0440             2201 SWITC=3                                                   572 KMCH
0441             2205 IF(WORKM.GT.NONDS)GO TO 2210                              573 KMCH
                C IF THIS IS A NODE*                                            574 KMCH
0442                  NDLBL(WORKM)=SWITC                                        575 KMCH
0443                  INBLS(WORKM)=0                                            576 KMCH
0444                  DISTND(WORKM)=0                                           577 KMCH
0445                  IF(WORKM.EQ.NEREND)GO TO 2450                             578 KMCH
0446                  IBW=BLSND(WORKM)                                          579 KMCH
0447                  IF(IBW.GT.NONDS)GO TO 2203                                580 KMCH
                C IF NEXT IS A NODE*                                            581 KMCH
0448                  TRND(IBW)=WORKM                                           582 KMCH
0449                  TREDG(IBW)=BLSEDG(WORKM)                                  583 KMCH
0450                  GO TO 2204                                                584 KMCH
                C IF NEXT IS A BLOSSOM                                          585 KMCH
0451             2203 ID=DNODE(IBW)                                             586 KMCH
0452                  TRND(ID)=WORKM                                            587 KMCH
0453                  TREDG(ID)=BLSEDG(WORKM)                                   588 KMCH
0454             2204 WORKM=BLSND(WORKM)                                        589 KMCH
0455                  IF(SWITC.EQ.2)GO TO 2206                                  590 KMCH
0456                  SWITC=2                                                   591 KMCH
0457                  GO TO 2205                                                592 KMCH
0458             2206 SWITC=3                                                   593 KMCH
0459                  GO TO 2205                                                594 KMCH
                C IF THIS ELEMENT IS A BLOSSOM                                  595 KMCH
0460             2210 DO 2211 KK=1,NONDS                                        596 KMCH
0461                  IF(OUTBLS(KK).NE.WORKM)GO TO 2211                         597 KMCH
0462                  NDLBL(KK)=SWITC                                           598 KMCH
0463                  DISTND(KK)=DNODE(WORKM)                                   599 KMCH
0464             2211 CONTINUE                                                  600 KMCH
0465                  PARORD(WORKM)=0                                           601 KMCH
0466                  BLSLBL(WORKM)=SWITC                                       602 KMCH
```

```
0467              IF(WORKM.EQ.NEREND)GO TO 2450                                  603 KMCH
0468              IBW=BLSEDG(WORKM)                                              604 KMCH
0469              IB=I(IBW)                                                      605 KMCH
0470              IF(OUTBLS(IB).EQ.WORKM)GO TO 2215                              606 KMCH
0471              WORKQ=J(IBW)                                                   607 KMCH
0472              GO TO 2216                                                     608 KMCH
0473           2215 WORKQ=IB                                                     609 KMCH
0474           2216 IBW=BLSND(WORKM)                                             610 KMCH
0475              ID=DNODE(IBW)                                                  611 KMCH
0476              IF(IBW.GT.NONDS)GO TO 2218                                     612 KMCH
                C IF NEXT IS A NODE*                                             613 KMCH
0477              TRND(IBW)=WORKQ                                                614 KMCH
0478              TREDG(IBW)=BLSEDG(WORKM)                                       615 KMCH
0479              GO TO 2204                                                     616 KMCH
                C IF NEXT IS A BLOSSOM*                                          617 KMCH
0480           2218 TRND(ID)=WORKQ                                               618 KMCH
0481              TREDG(ID)=BLSEDG(WORKM)                                        619 KMCH
0482              GO TO 2204                                                     620 KMCH
                C ENDING ROUTINE                                                 621 KMCH
0483           2450 BLSLBL(BLOBLS)=0                                             622 KMCH
0484              PARORD(BLOBLS)=-1                                              623 KMCH
0485              DNODE(BLOBLS)=0                                                624 KMCH
0486              WORKQ=NEREND                                                   625 KMCH
0487           2451 WORKM=BLSND(WORKQ)                                           626 KMCH
0488              BLSND(WORKQ)=0                                                 627 KMCH
0489              BLSEDG(WORKQ)=0                                                628 KMCH
0490              WORKQ=WORKM                                                    629 KMCH
0491              IF(WORKQ.EQ.NEREND)GO TO 88                                    630 KMCH
0492              GO TO 2451                                                     631 KMCH
0493            610 RETURN                                                       634RKMCH
0494              END
```

```
0001              SUBROUTINE OUTPUT
0002              REAL NDWT(70),LAMDA,EDGWT     ,Z(70),MCHWT
0003              INTEGER MATCH(1080),HIHEST, TRND(46),
                 1TREDG(46),NDLBL(70),OUTBLS(46),INBLS(70),
                 2ROOT(70),BLSND(70),
                 3BLSEDG(70),DISTND(46),HIBLS,BLSLBL(70),
                 4PARORD(70),DNODE(70),I     ,J,OUTMCH
0004              INTEGER STACK(46),OSTAR,WRKBLS,          DISTEL,
                 1 CUREL,TEST,SAVND,SCARD,BCARD                                  009 OTPT
0005              EQUIVALENCE(Z,NDWT),(NDLBL,BLSLBL),(INBLS,PARORD),              010 OTPT
                 1  (ROOT,DNODE)                                                 011 OTPT
0006              COMMON/KMOTPT/  I(1080),  J(1080),NONDS,NOEDGS,OUTMCH(1080),
                 1 EDGWT(1080)
0007              COMMON /OTPT/ SCALAR,      NDWT,       MATCH,                  012ROTPT
                 1 TOTEWT,K,HIHEST,TRND,TREDG,NDLBL,                             013 OTPT
                 2 OUTBLS,INBLS,ROOT,BLSND,BLSEDG,DISTND,LOBLS,HIBLS,            014 OTPT
                 3      IN,IOUT                                                  015ROTPT
                C                                                                016 OTPT
                C CALCULATE LAMDA                                                017 OTPT
                C                                                                018 OTPT
0008              LAMDA=2.*SCALAR                                                019 OTPT
0009              DO 7000 KK=1,NOEDGS                                            020 OTPT
0010           7000 OUTMCH(KK)=MATCH(KK)                                         021 OTPT
                C                                                                022 OTPT
                C                                                                023 OTPT
                C FIND DUAL                                                      024 OTPT
                C                                                                025 OTPT
0011              DUAL=0.                                                        026 OTPT
                C                                                                027 OTPT
                C FIRST PART OF DUAL CALCULATION STARTS HERE*                    028 OTPT
0012              DO 3412 KK=1,NONDS                                             029 OTPT
0013            3412 DUAL=DUAL+(NDWT(KK)-(LAMDA/2.))                             030 OTPT
                C                                                                031 OTPT
                C SECOND PART OF DUAL CALCULATION (INVOLVING S SUB P             032 OTPT
                C (AND R SUB P) AND B SUB P) STARTS HERE*                        033 OTPT
0014              DO 3417 KK=LOBLS,HIBLS                                         034 OTPT
0015              IF(PARORD(KK).EQ.-1)GO TO 3417                                 035 OTPT
0016              DO 3423 LA=1,NONDS                                             036 OTPT
0017            3423 STACK(LA)=0                                                 037 OTPT
0018              DO 3418 KKE=1,NOEDGS                                           038 OTPT
0019              IK=I(KKE)                                                      039 OTPT
0020              JK=J(KKE)                                                      040 OTPT
0021              IF(OUTBLS(IK).EQ.0)GO TO 3420                                  041 OTPT
0022              OSTAR=INBLS(IK)                                                042 OTPT
```

```
0467              IF(WORKM.EQ.NEREND)GO TO 2450                                  603 KMCH
0468              IBW=BLSEDG(WORKM)                                              604 KMCH
0469              IB=I(IBW)                                                      605 KMCH
0470              IF(OUTBLS(IB).EQ.WORKM)GO TO 2215                              606 KMCH
0471              WORKQ=J(IBW)                                                   607 KMCH
0472              GO TO 2216                                                     608 KMCH
0473           2215 WORKQ=IB                                                     609 KMCH
0474           2216 IBW=BLSND(WORKM)                                             610 KMCH
0475              ID=DNODE(IBW)                                                  611 KMCH
0476              IF(IBW.GT.NONDS)GO TO 2218                                     612 KMCH
                C IF NEXT IS A NODE*                                             613 KMCH
0477              TRND(IBW)=WORKQ                                                614 KMCH
0478              TREDG(IBW)=BLSEDG(WORKM)                                       615 KMCH
0479              GO TO 2204                                                     616 KMCH
                C IF NEXT IS A BLOSSOM*                                          617 KMCH
0480           2218 TRND(ID)=WORKQ                                               618 KMCH
0481              TREDG(ID)=BLSEDG(WORKM)                                        619 KMCH
0482              GO TO 2204                                                     620 KMCH
                C ENDING ROUTINE                                                 621 KMCH
0483           2450 BLSLBL(BLOBLS)=0                                             622 KMCH
0484              PARORD(BLOBLS)=-1                                              623 KMCH
0485              DNODE(BLOBLS)=0                                                624 KMCH
0486              WORKQ=NEREND                                                   625 KMCH
0487           2451 WORKM=BLSND(WORKQ)                                           626 KMCH
0488              BLSND(WORKQ)=0                                                 627 KMCH
0489              BLSEDG(WORKQ)=0                                                628 KMCH
0490              WORKQ=WORKM                                                    629 KMCH
0491              IF(WORKQ.EQ.NEREND)GO TO 88                                    630 KMCH
0492              GO TO 2451                                                     631 KMCH
0493            610 RETURN                                                       634RKMCH
0494              END
```

```
0001              SUBROUTINE OUTPUT
0002              REAL NDWT(70),LAMDA,EDGWT    ,Z(70),MCHWT
0003              INTEGER MATCH(1080),HIHEST, TRND(46),
                 1TREDG(46),NDLBL(70),OUTBLS(46),INBLS(70),
                 2ROOT(70),BLSND(70),
                 3BLSEDG(70),DISTND(46),HIBLS,BLSLBL(70),
                 4PARORD(70),DNODE(70),I    ,J,OUTMCH
0004              INTEGER STACK(46),DSTAR,WRKBLS,          DISTEL,
                 1 CUREL,TEST,SAVND,SCARD,BCARD                                  009 OTPT
0005              EQUIVALENCE(Z,NDWT),(NDLBL,BLSLBL),(INBLS,PARORD),              010 OTPT
                 1  (ROOT,DNODE)                                                 011 OTPT
0006              COMMON/KMOTPT/  I(1080),  J(1080),NONDS,NOEDGS,OUTMCH(1080),
                 1 EDGWT(1080)
0007              COMMON /OTPT/ SCALAR,      NDWT,       MATCH,                012ROTPT
                 1 TOTEWT,X,HIHEST,TRND,TREDG,NDLBL,                            013 OTPT
                 2 OUTBLS,INBLS,ROOT,BLSND,BLSEDG,DISTND,LOBLS,HIBLS,           014 OTPT
                 3     IN,IOUT                                                  015ROTPT
                C                                                               016 OTPT
                C CALCULATE LAMDA                                               017 OTPT
                C                                                               018 OTPT
0008              LAMDA=2.*SCALAR                                               019 OTPT
0009              DO 7000 KK=1,NOEDGS                                           020 OTPT
0010           7000 OUTMCH(KK)=MATCH(KK)                                        021 OTPT
                C                                                               022 OTPT
                C                                                               023 OTPT
                C FIND DUAL                                                     024 OTPT
                C                                                               025 OTPT
0011              DUAL=0.                                                       026 OTPT
                C                                                               027 OTPT
                C FIRST PART OF DUAL CALCULATION STARTS HERE*                   028 OTPT
0012              DO 3412 KK=1,NONDS                                            029 OTPT
0013           3412 DUAL=DUAL+(NDWT(KK)-(LAMDA/2.))                             030 OTPT
                C                                                               031 OTPT
                C SECOND PART OF DUAL CALCULATION (INVOLVING S SUB P            032 OTPT
                C (AND R SUB P) AND B SUB P) STARTS HERE*                       033 OTPT
0014              DO 3417 KK=LOBLS,HIBLS                                        034 OTPT
0015              IF(PARORD(KK).EQ.-1)GO TO 3417                                035 OTPT
0016              DO 3423 LA=1,NONDS                                            036 OTPT
0017           3423 STACK(LA)=0                                                 037 OTPT
0018              DO 3418 KKE=1,NOEDGS                                          038 OTPT
0019              IK=I(KKE)                                                     039 OTPT
0020              JK=J(KKE)                                                     040 OTPT
0021              IF(OUTBLS(IK).EQ.0)GO TO 3420                                 041 OTPT
0022              DSTAR=INBLS(IK)                                               042 OTPT
```

```
0023          4549 IF(DSTAR.LE.0)GO TO 3420                                    043 OTPT
0024               IF(DSTAR.EQ.KK)GO TO 4558                                   044 OTPT
0025               DSTAR=PARORD(DSTAR)                                         045 OTPT
0026               GO TO 4549                                                  046 OTPT
0027          4558 STACK(IK)=1                                                 047 OTPT
0028          3420 IF(OUTBLS(JK).EQ.0)GO TO 3418                               048 OTPT
0029               DSTAR=INBLS(JK)                                             049 OTPT
0030          4542 IF(DSTAR.LE.0)GO TO 3418                                    050 OTPT
0031               IF(DSTAR.EQ.KK)GO TO 4291                                   051 OTPT
0032               DSTAR=PARORD(DSTAR)                                         052 OTPT
0033               GO TO 4542                                                  053 OTPT
0034          4291 STACK(JK)=1                                                 054 OTPT
0035          3418 CONTINUE                                                    055 OTPT
0036               SCARD=0                                                     056 OTPT
0037               DO 3426 LA=1,NONDS                                          057 OTPT
0038          3426 SCARD=SCARD+STACK(LA)                                       058 OTPT
0039               TCARD=SCARD-1                                               059 OTPT
0040               DUAL=DUAL+TCARD/2.*Z(KK)                                    060 OTPT
0041          3417 CONTINUE                                                    061 OTPT
              C THIRD NO FOURTH PARTS OF DUAL CALCULATION STARTS HERE*         062 OTPT
0042               A=K                                                         063 OTPT
0043               DUAL = DUAL+A*LAMDA                                         064 OTPT
              C                                                                065 OTPT
              C                                                                066 OTPT
              C START INDUCTION                                                067 OTPT
              C                                                                068 OTPT
              C LOAD OUTERMOST BLOSSOMS INTO STACK                             069 OTPT
0044               MARKER=0                                                    070 OTPT
0045               DO 1500 KK=LOBLS,HIBLS                                      071 OTPT
0046               IF(PARORD(KK).NE.0)GO TO 1500                               072 OTPT
0047               MARKER=MARKER+1                                             073 OTPT
0048               STACK(MARKER)=-KK                                           074 OTPT
0049          1500 CONTINUE                                                    075 OTPT
              C                                                                076 OTPT
              C PULL BLOSSOM AT MARKER                                         077 OTPT
0050          1502 IF(MARKER.EQ.0)GO TO 2632                                   078 OTPT
0051               IF(STACK(MARKER).LT.0)GO TO 1503                            079 OTPT
0052               WRKBLS=STACK(MARKER)                                        080 OTPT
0053               STACK(MARKER)=0                                             081 OTPT
0054               MARKER=MARKER-1                                             082 OTPT
0055               GO TO 1510                                                  083 OTPT
0056          1503 WRKBLS=-STACK(MARKER)                                       084 OTPT
0057               STACK(MARKER)=0                                             085 OTPT
0058               MARKER=MARKER-1                                             086 OTPT
0059               DSTAR=DNODE(WRKBLS)                                         087 OTPT
              C FIND DISTINGUISHED ELEMENT                                     088 OTPT
0060          1510 DISTEL=DNODE(WRKBLS)                                        089 OTPT
0061               IF(INBLS(DISTEL).EQ.WRKBLS)GO TO 1701                       090 OTPT
0062               DISTEL=INBLS(DISTEL)                                        091 OTPT
0063          1518 IF(PARORD(DISTEL).EQ.WRKBLS)GO TO 1701                      092 OTPT
0064               DISTEL=PARORD(DISTEL)                                       093 OTPT
0065               GO TO 1518                                                  094 OTPT
0066          1701 IF(DISTEL.LE.NONDS)GO TO 1540                               095 OTPT
0067               IP=PARORD(DISTEL)                                           096 OTPT
0068               DNODE(DISTEL)=DNODE(IP)                                     097 OTPT
0069               MARKER=MARKER+1                                             098 OTPT
0070               STACK(MARKER)=DISTEL                                        099 OTPT
0071          1540 CUREL=BLSND(DISTEL)                                         100 OTPT
0072               GO TO 1555                                                  101 OTPT
              C COMMON INDUCTION ROUTINE                                       102 OTPT
              C STARTING HERE WE TREAT A VERTEX FOLLOWED BY                    103 OTPT
              C A SOLID EDGE                                                   104 OTPT
0073          1555 IB=BLSEDG(CUREL)                                            105 OTPT
0074               IK=I(IB)                                                    106 OTPT
0075               JK=J(IB)                                                    107 OTPT
0076               OUTMCH(IB)=1                                                108 OTPT
0077               IF(CUREL.LE.NONDS)GO TO 1558                                109 OTPT
              C IF THIS ELEMENT IS A BLOSSOM                                   110 OTPT
0078               TEST=INBLS(IK)                                              111 OTPT
0079          1560 IF(TEST.EQ.CUREL)GO TO 1563                                 112 OTPT
0080               IF(TEST.EQ.0)GO TO 1566                                     113 OTPT
0081               TEST=PARORD(TEST)                                           114 OTPT
0082               GO TO 1560                                                  115 OTPT
0083          1563 DNODE(CUREL)=IK                                             116 OTPT
0084               SAVND=JK                                                    117 OTPT
0085               GO TO 1570                                                  118 OTPT
0086          1566 DNODE(CUREL)=JK                                             119 OTPT
0087               SAVND=IK                                                    120 OTPT
0088          1570 MARKER=MARKER+1                                             121 OTPT
0089               STACK(MARKER)=CUREL                                         122 OTPT
0090               GO TO 1590                                                  123 OTPT
```

```
0023          4549 IF(DSTAR.LE.0)GO TO 3420
0024               IF(DSTAR.EQ.KK)GO TO 4558
0025               DSTAR=PARORD(DSTAR)
0026               GO TO 4549
0027          4558 STACK(IK)=1
0028          3420 IF(OUTBLS(JK).EQ.0)GO TO 3418
0029               DSTAR=INBLS(JK)
0030          4542 IF(DSTAR.LE.0)GO TO 3418
0031               IF(DSTAR.EQ.KK)GO TO 4291
0032               DSTAR=PARORD(DSTAR)
0033               GO TO 4542
0034          4291 STACK(JK)=1
0035          3418 CONTINUE
0036               SCARD=0
0037               DO 3426 LA=1,NONDS
0038          3426 SCARD=SCARD+STACK(LA)
0039               TCARD=SCARD-1
0040               DUAL=DUAL+TCARD/2.*Z(KK)
0041          3417 CONTINUE
              C THIRD NO FOURTH PARTS OF DUAL CALCULATION STARTS HERE*
0042               A=K
0043               DUAL = DUAL+A*LAMDA
              C
              C
              C START INDUCTION
              C
              C LOAD OUTERMOST BLOSSOMS INTO STACK
0044               MARKER=0
0045               DO 1500 KK=LOBLS,HIBLS
0046               IF(PARORD(KK).NE.0)GO TO 1500
0047               MARKER=MARKER+1
0048               STACK(MARKER)=-KK
0049           1500 CONTINUE
              C
              C PULL BLOSSOM AT MARKER
0050           1502 IF(MARKER.EQ.0)GO TO 2632
0051               IF(STACK(MARKER).LT.0)GO TO 1503
0052               WRKBLS=STACK(MARKER)
0053               STACK(MARKER)=0
0054               MARKER=MARKER-1
0055               GO TO 1510
0056           1503 WRKBLS=-STACK(MARKER)
0057               STACK(MARKER)=0
0058               MARKER=MARKER-1
0059               DSTAR=DNODE(WRKBLS)
              C FIND DISTINGUISHED ELEMENT
0060           1510 DISTEL=DNODE(WRKBLS)
0061               IF(INBLS(DISTEL).EQ.WRKBLS)GO TO 1701
0062               DISTEL=INBLS(DISTEL)
0063           1518 IF(PARORD(DISTEL).EQ.WRKBLS)GO TO 1701
0064               DISTEL=PARORD(DISTEL)
0065               GO TO 1518
0066           1701 IF(DISTEL.LE.NONDS)GO TO 1540
0067               IP=PARORD(DISTEL)
0068               DNODE(DISTEL)=DNODE(IP)
0069               MARKER=MARKER+1
0070               STACK(MARKER)=DISTEL
0071           1540 CUREL=BLSND(DISTEL)
0072               GO TO 1555
              C COMMON INDUCTION ROUTINE
              C STARTING HERE WE TREAT A VERTEX FOLLOWED BY
              C A SOLID EDGE
0073           1555 IB=BLSEDG(CUREL)
0074               IK=I(IB)
0075               JK=J(IB)
0076               OUTMCH(IB)=1
0077               IF(CUREL.LE.NONDS)GO TO 1558
              C IF THIS ELEMENT IS A BLOSSOM
0078               TEST=INBLS(IK)
0079           1560 IF(TEST.EQ.CUREL)GO TO 1563
0080               IF(TEST.EQ.0)GO TO 1566
0081               TEST=PARORD(TEST)
0082               GO TO 1560
0083           1563 DNODE(CUREL)=IK
0084               SAVND=JK
0085               GO TO 1570
0086           1566 DNODE(CUREL)=JK
0087               SAVND=IK
0088           1570 MARKER=MARKER+1
0089               STACK(MARKER)=CUREL
0090               GO TO 1590
```

```
043 OTPT
044 OTPT
045 OTPT
046 OTPT
047 OTPT
048 OTPT
049 OTPT
050 OTPT
051 OTPT
052 OTPT
053 OTPT
054 OTPT
055 OTPT
056 OTPT
057 OTPT
058 OTPT
059 OTPT
060 OTPT
061 OTPT
062 OTPT
063 OTPT
064 OTPT
065 OTPT
066 OTPT
067 OTPT
068 OTPT
069 OTPT
070 OTPT
071 OTPT
072 OTPT
073 OTPT
074 OTPT
075 OTPT
076 OTPT
077 OTPT
078 OTPT
079 OTPT
080 OTPT
081 OTPT
082 OTPT
083 OTPT
084 OTPT
085 OTPT
086 OTPT
087 OTPT
088 OTPT
089 OTPT
090 OTPT
091 OTPT
092 OTPT
093 OTPT
094 OTPT
095 OTPT
096 OTPT
097 OTPT
098 OTPT
099 OTPT
100 OTPT
101 OTPT
102 OTPT
103 OTPT
104 OTPT
105 OTPT
106 OTPT
107 OTPT
108 OTPT
109 OTPT
110 OTPT
111 OTPT
112 OTPT
113 OTPT
114 OTPT
115 OTPT
116 OTPT
117 OTPT
118 OTPT
119 OTPT
120 OTPT
121 OTPT
122 OTPT
123 OTPT
```

```
                C IF THIS ELEMENT IS A NODE                                          124 OTPT
0091             1558 IF(IK.EQ.CUREL)GO TO 1601                                      125 OTPT
0092                  SAVND=IK                                                       126 OTPT
0093                  GO TO 1590                                                     127 OTPT
0094             1601 SAVND=JK                                                       128 OTPT
0095             1590 IF(BLSND(CUREL).EQ.DISTEL)GO TO 1502                           129 OTPT
0096                  CUREL=BLSND(CUREL)                                             130 OTPT
                C STARTING HERE WE TREAT A VERTEX FOLLOWED BY A DOTTED EDG           131 OTPT
                C                                                                    132 OTPT
0097                  IF(CUREL.LE.NONDS)GO TO 1603                                   133 OTPT
                C IF THIS ELEMENT IS A BLOSSOM                                       134 OTPT
0098                  DNODE(CUREL)=SAVND                                             135 OTPT
0099                  MARKER=MARKER+1                                                136 OTPT
0100                  STACK(MARKER)=CUREL                                            137 OTPT
0101             1603 IF(BLSND(CUREL).EQ.DISTEL)GO TO 1502                           138 OTPT
                C IS THIF THIS ELEMENT IS A NODE                                     139 OTPT
0102                  CUREL=BLSND(CUREL)                                             140 OTPT
0103                  GO TO 1555                                                     141 OTPT
                C CALCULATE THE MATCHING WEIGHT                                      142 OTPT
                C                                                                    143 OTPT
0104             2632 MCHWT=0.0                                                      144 OTPT
0105                  DO 261 KK=1,NOEDGS                                             145 OTPT
0106                  IF(OUTMCH(KK).EQ.0)GO TO 261                                   146 OTPT
0107                  MCHWT=MCHWT+EDGWT(KK)                                          147 OTPT
0108              261 CONTINUE                                                       148 OTPT
0109              295 RETURN                                                         149 OTPT
0110                  END
```

```
0001                  SUBROUTINE ARMS
                C
                C
                C                 ARMS  SEQUENTIALLY FINDS THE LEAST COST SET OF ARMS
                C
0002                  COMMON DIST(90,90),IPATH(90,90),JPATH(90),IEND(270),
                     1JEND(270),COST(270),NODD(90),ELOAD(270),NSOURC,NTERM,INUM,
                     2 NUMNDS,NUMEGS,IINODE(1080),JJNODE(1080),AROUND(90,10),NUMINC(90),
                     3 INNODE(60,10),CYCNDS(60),SP(60,60),NRSTND(60,60),CLOAD(60),
                     4 DLOAD(20),IN(20,40),NODE(90),NO(20),JX(60,60),ITIE(100),NF(100),
                     5 SPB(60),M,N,TCOST,INF,FINF,NCYNDS,IDEPOT,AVLOAD,C,NATCH,
                     6 NEXTND(10),ITOUR(20,300),NOTOUR(20) ,STOP,IMOUT,COSTK(20)
0003                  COMMON/KMOTPT/IKM(1080),JKM(1080),NONDS,NOEDGS, MATCH(1080),
                     1 EDGWT(1080)
0004                  COMMON/BILD/INTREE (50,20),IRETRN,NUMBER,IRMS ,MBEGIN,IBEGIN,
                     1 IONE,ALIST(200),NEXT(200),IPACK(200,2), MASK(30),INJX(60),IDONE
0005                  INTEGER MBEGIN
                C
                C                 BYPASS INITIALIZATION IF ALREADY DONE
                C
0006                  IF(IRETRN .EQ. 1) GO TO 140
                C
                C                 INITIALIZATION
                C
0007                  ALIST(1) = 0.
0008                  M1 = M + 1
0009                  DO 100 I = 2,M1
0010             100 ALIST(1) = ALIST(1) + SP(1,NODE(I))
0011                  NEXT(1) = 0
0012                  ALIST(2) = 0.
0013                  NEXT(2) = 3
0014                  DO 110 I = 3,200
0015                  ALIST(I) = 0.
0016                  NEXT(I) = I + 1
0017                  DO 110 J = 1,2
0018             110 IPACK(I,J) = 0
0019                  NEXT (200) = 0
0020                  IPACK(2,1) =0
0021                  IPACK(2,2) = 0
0022                  IONE = 0
0023                  DO 120 I = 1,30
0024                  MASK(I) = 2**(I - 1)
0025             120 IONE = IONE + MASK(I)
0026                  IPACK(1,1) = 0
0027                  IPACK(1,2) = 0
0028                  DO 130 I = 2,M1
0029             130 IPACK (1,1) = IORZ(IPACK(1,1) ,MASK(I))
0030                  IBEGIN = 2
```

```
0031            MBEGIN = 1
0032            GO TO 275
                C
                C           DETERMINE THE MOST RECENT INFEASIBEL SET OF ARMS
                C
0033          140 J = 0
0034            DO 160 I = 1,30
0035            IF (IANDZ(IPACK(MBEGIN,1),MASK (I))  .GT. 0) GO TO 145
0036            NOTIN = I
0037            GOTO 150
0038          145  J = J+ 1
0039          150 IF(J .EQ. M) GO TO 190
0040          160 CONTINUE
0041            DO 180 I = 31,60
0042            L = I - 30
0043            IF ( IANDZ ((IPACK(MBEGIN , 2),MASK(L)) .GT. 0) GO TO 165
0044            NOTIN = I
0045            GO TO 170
0046          165 J = J + 1
0047          170 IF( J .EQ. M) GO TO 190
0048          180 CONTINUE
                C
                C           ADD A MORE EXPENSIVE ARM TO SET AND ELININATE ARMS ONE
                C           AT A TIME
                C
                C           IF I  = NCYNDS WE CANT BRANCH FURTHER FROM THIS NODE
                C
0049          190 IF(I .EQ. NCYNDS) GO TO 270
0050            NOT = NOTIN + 1
0051            DO 200 JJ = NOT,I
0052            ALIST(IBEGIN) = ALIST(MBEGIN) + SP(1,NODE(I + 1))- SP(1, NODE(JJ))
0053            IF(ALIST(IBEGIN) .GE. ALIST(MBEGIN)) GO TO 209
0054            ISAVE = NEXT(IBEGIN)
0055            NEXT (IBEGIN) = MBEGIN
0056            MBEGIN =IBEGIN
0057            GO TO 230
0058          209 IF(NEXT(MBEGIN) .NE. 0) GO TO 208
0059            NEXT(MBEGIN) = IBEGIN
0060            ISAVE = NEXT(IBEGIN)
0061            NEXT(IBEGIN) = 0
0062            GO TO 230
0063          208 II  = NEXT(MBEGIN)
0064            LAST = MBEGIN
0065          210 IF(ALIST(IBEGIN) .LE. ALIST(II)) GO TO 220
0066            LAST  = II
0067            II = NEXT (II)
0068            IF(II .NE. 0) GO TO 210
0069          220 NEXT(LAST) = IBEGIN
0070            ISAVE  = NEXT(IBEGIN)
0071            NEXT(IBEGIN) = II
                C
                C           CALCULATE IPACK
                C
0072          230 IF(JJ .GT. 30 ) GO TO 250
0073            ITEMP = IXORZ (IONE, MASK(JJ))
0074            JTEMP  = IANDZ(ITEMP ,IPACK(MBEGIN,1))
0075            IF(I+1.GT.30 ) GO TO 240
0076            IPACK(IBEGIN,1) = IORZ(JTEMP,MASK(I + 1))
0077            GO TO 260
0078          240 L = I - 30
0079            IPACK(IBEGIN,1) = JTEMP
0080            IPACK(IBEGIN,2) = IORZ(IPACK(MBEGIN,2), MASK(L + 1) )
0081            GO TO 260
0082          250 IJ = JJ - 30
0083            ITEMP = IXORZ (IONE, MASK(IJ))
0084            L = I - 30
0085            IPACK(IBEGIN,1) = IPACK(MBEGIN,1)
0086            JTEMP = IANDZ(ITEMP,IPACK(MBEGIN,2))
0087            IPACK(IBEGIN,2) = IORZ(JTEMP,MASK(L + 1))
0088          260 IBEGIN = ISAVE
0089          200 CONTINUE
                C
                C           RECALCULATE LIST INDICIES AND REMOVE
                C           INFEASIBLE SET FROM ACTIVE LIST
                C
0090          270 ALIST(MBEGIN) = 0.
0091            IPACK(MBEGIN,1) = 0
0092            IPACK(MBEGIN,2) = 0
0093            IBEGIN = MBEGIN
0094            MBEGIN = NEXT(MBEGIN    )
0095            NEXT(IBEGIN) = ISAVE
```

```
C
C               SET ARMS
C
0096            IF(ALIST(HBEGIN) .LT. STOP) GO TO 275
0097            STOP = - 2.
0098            WRITE(6,273)
0099          273 FORMAT(////5X, 'THE COMPUTER CODE CANNOT GENERATE A SOLUTION
                1 BETTER  THAN THE TERMINATION COST WHICH WAS SUBMITTED
                2 BY THE USER AS PART OF THE DATA'/////)
0100            GO TO 340
0101          275 J = 0
0102            DO 290 I = 1,30
0103            IF(IANDZ(IPACK(HBEGIN,1), MASK(I)) .LE. 0) GO TO 280
0104            J = J + 1
0105            IN(J,1) = NODE(I)
0106          280 IF(J .EQ. M) GO TO 320
0107          290 CONTINUE
0108            DO 310 I = 1,30
0109            IF( IANDZ(IPACK(HBEGIN,2), MASK(I)) .LE. 0) GO TO  300
0110            J = J + 1
0111            IN(J,1) = NODE(30 + I)
0112          300 IF(J .EQ. M) GO TO 320
0113          310 CONTINUE
0114          320 DO 330 K = 1,M
0115            JX(1,IN(K,1)) = K
0116            JX(IN(K,1) ,1) = K
0117          330 NO(K) = 1
0118            IF(IMOUT .NE. 1) GO TO 340
0119            WRITE (6,8001) (IN(K,1) ,K = 1,M)
0120         8001 FORMAT(5X,'ARMS ARE ', 10I5)
0121          340 RETURN
                END



0001            SUBROUTINE    SCREEN
C
C               SCREEN  DETERMINES IF THE ARMS ARE INFEASIBLE
C               ACCORDING TO CERTAIN RULES
C
0002            COMMON DIST(90,90),IPATH(90,90),JPATH(90),IEND(270),
                1JEND(270),COST(270),NODD(90),ELOAD(270),NSOURC,NTERM,INUM,
                2 NUMNDS,NUMEGS,IINODE(1080),JJNODE(1080),AROUND(90,10),NUMINC(90),
                3 INNODE(60,10),CYCNDS(60),SP(60,60),NRSTND(60,60),CLOAD(60),
                4 DLOAD(20),IN(20,40),NODE(90),NO(20),JX(60,60),ITIE(100),NF(100),
                5 SPB(60),M,N,TCOST,INF,FINF,NCYNDS,IDEPOT,AVLOAD,C,NATCH,
                6 NEXTND(10),ITOUR(20,300),NOTOUR(20) ,STOP,IMOUT,COSTK(20)
0003            COMMON/KNOTPT/IKM(1080),JKM(1080),NONDS,NOEDGS, MATCH(1090),
                1 EDGWT(1080)
0004            COMMON/BILD/INTREE (50,20),IRETRN,NUMBER,IRMS ,HBEGIN,IBEGIN,
                1 IONE,ALIST(200),NEXT(200),IPACK(200,2), MASK(30),INJX(60),IDONE
0005            INTEGER HBEGIN
0006          100 IRETRN = 0
0007            CALL PATHS
C
C               INITIALIZE INTREE
C
0008            DO 120 K = 1,M
0009            DO 110 I = 1,NCYNDS
0010            IF(INTREE(I,K) .GT. 1) GO TO 110
0011            INTREE(I,K) = 0
0012          110 CONTINUE
0013          120 INTREE(IN(K,1) ,K) =1
C
C               LABEL NODES WITHIN REACH OF TREES AND DETERMINE
C               IF THERE ARE ANY ISOLATED REGIONS
C
0014            DO 140 K = 1,M
0015            COUNT = DLOAD(K)
0016            DO 150 I = 2,NCYNDS
0017            NOK = NO(K)
0018            DO 145 JI = 1,NOK
0019            IF(DIST(I,IN(K,JI))+ DLOAD(K) .GT. C) GO TO 145
0020            GO TO 147
0021          145 CONTINUE
0022            GO TO 150
0023          147 COUNT = COUNT + CLOAD(I)
0024            INTREE (I,K) = 1
```

```
0025          150 CONTINUE
0026              IF (COUNT .LE. C - M*(C - AVLOAD)) GO TO 340
0027          140 CONTINUE
              C
              C               DETERMINE IF A NODE ISNT WITHIN REACH OF ANY ARM
              C
0028              DO 335 I = 2,NCYNDS
0029              KOUNT = 0
0030              DO 170 K = 1,M
0031          170 KOUNT = INTREE(I,K) + KOUNT
0032              IF(KOUNT .NE. 0) GO TO 335
0033              IF(JPATH(1) .EQ. 0) GO TO 340
              C
              C               FROM HERE TO LINE 335 WE ARE CHECKING TO SEE IF IT IS
              C               POSSIBLE TO BRING THE UNREACHABLE NODES INTO THE TREE BY
              C               REARRANGING THE SUBTREE BRANCHES
              C
              C               CHECK IF ADDITION OF NODE IN QUESTION TO TREE IS
              C               FEASIBLE
              C
0034              DO 215 K = 1,M
0035              NOK = NO(K)
0036              DO 210 II = 1,NOK
0037              IF(DIST(I,IN(K,II))+DLOAD(K).LE. C) GO TO 220
0038          210 CONTINUE
0039              GO TO 250
0040          215 CONTINUE
              C
              C               IF FEASIBLE ADD NODE
              C
0041          220 NSOURC = I
0042              NTERM = IN(K,II)
0043              CALL DECODE
0044              INUM1 = INUM - 1
0045              DO 230 II = 1,INUM1
0046              IN(K,NO(K)  + II) = JPATH(INUM -II)
0047              JX(JPATH(INUM - II + 1),JPATH(INUM -I)) = K
0048              JX(JPATH(INUM - II),JPATH(INUM - II + 1)) = K
0049              INTREE(JPATH(INUM - II),K) = 2
0050              DO 240 JJ = 1,NCYNDS
0051          240 DIST(JPATH(INUM -II),JJ) = FINF
0052          230 CONTINUE
0053              NO(K) = NO(K) + INUM -1
0054              NUMBER = NUMBER + INUM -1
0055              GO TO 100
              C
              C               ELIMINATE OLD SUBTREE AND CHECK TO SEE IF ADDITION OF
              C               FURTHEST NODE TO NODE IN QUESTION IS FEASIBLE
              C
              C               DETERMINE THE SUBTREE WHICH IS ISOLATING THE NODE IN
              C               QUESTION
              C
0056          250 DO 180 K = 1,M
0057              IF(INTREE (JPATH(1),K) .GT. 0) GO TO 190
0058          180 CONTINUE
0059          190 NOK = NO(K)
0060              NUMBER = NUMBER - NOK + 1
0061              DO 260 J = 2,NOK
0062          260 IN(K,J) = 0
0063              NO(K) = 1
0064              DO 270 II= 2,NCYNDS
0065              DO 270 J = 2,NCYNDS
0066              IF(JX(II,J).NE. K) GO TO 270
0067              JX(II,J)= 0
0068              JX(J,II)= 0
0069          270 CONTINUE
0070              KEND = JPATH(1)
0071              NSOURC = I
0072              NTERM = IN(K,1)
0073              CALL DECODE
              C
              C               CHECK TO SEE IF ANY SUBTREES CAN BE  FORMED ON THE
              C               NODES WITHOUT VIOLATING FEASIBILITY
              C
0074              INUM1 = INUM - 1
0075              DO 280 II=  1,INUM1
0076              IF(DIST(KEND,JPATH(II)) + DIST(JPATH(1),IN(K,1)) + CLOAD(IN(K,1))
                 C  .LE. C) GO TO 290
0077          280 CONTINUE
0078              GO TO 340
              C
```

```
C            IF FEASIBLE PUT THE PROPER EDGES IN SUBTREE
C
0079      290 DO 300 III = 1,INUM1
0080          IN(K,III + 1) = JPATH(INUM - III)
0081          INTREE (JPATH(INUM - III), K) = 2
0082          JX(JPATH(INUM - III + 1),JPATH(INUM - III) ) = K
0083          JX(JPATH(INUM - III) , JPATH(INUM - III + 1)) = K
0084          DO 310 JJ = 1,NCYNDS
0085      310 DIST(JPATH(INUM - III),JJ ) = FINF
0086      300 CONTINUE
0087          NO(K) = NO(K) + INUM - 1
0088          NUMBER  = NUMBER + INUM - 1
0089          NSOURC = KEND
0090          NTERM = JPATH(II)
0091          CALL DECODE
0092          INUM1 = INUM - 1
0093          DO 320 III = 1,INUM1
0094          IN(K,NO(K) + III) = JPATH(INUM - III)
0095          INTREE(JPATH(INUM - III),K) = 2
0096           JX(JPATH (INUM - III), JPATH(INUM - III + 1)) = K
0097          DO 330 JJ = 1,NCYNDS
0098      330 DIST(JPATH(INUM - III) ,JJ) =FINF
0099      320 CONTINUE
0100          NO(K) = NO(K) + INUM - 1
0101          NUMBER = NUMBER + INUM - 1
0102          GO TO 100
0103      335 CONTINUE
0104          GO TO 350
0105      340 IRETRN = 1
0106      350 RETURN
0107          END



0001          SUBROUTINE BUILD
C
C                  BUILD  ADDS CYCLE NODES TO SUBTREES
C
0002           COMMON DIST(90,90),IPATH(90,90),JPATH(90),IEND(270),
              1JEND(270),COST(270),NODD(90),ELOAD(270),NSOURC,NTERM,INUM,
              2 NUMNDS,NUMEGS,IINODE(1080),JJNODE(1080),AROUND(90,10),NUMINC(90),
              3 INNODE(60,10),CYCNDS(60),SP(60,60),NRSTND(60,60),CLOAD(60),
              4 DLOAD(20),IN(20,40),NODE(90),NO(20),JX(60,60),ITIE(100),VF(100),
              5 SPB(60),M,N,TCOST,INF,FINF,NCYNDS,IDEPOT,AVLOAD,C,NATCH,
              6 NEXTND(10),ITOUR(20,300),NOTOUR(20) ,STOP,IMOUT,COSTK(20)
0003           COMMON/KMOTPT/IKM(1080),JKM(1080),NONDS,NOEDGS, MATCH(1080),
              1 EDGWT(1080)
0004           COMMON/BILD/INTREE (50,20),IRETRN,NUMBER,IRMS ,HBEGIN,IBEGIN,
              1 IONE,ALIST(200),NEXT(200),IPACK(200,2), MASK(30),INJX(60),IDONE
0005           INTEGER HBEGIN
0006           DIMENSION IOUT(20)
0007           DO 90 I = 1,M
0008           NO(I) = 1
0009       90  IOUT(I) = 0
0010           NUMBER = M + 1
0011           LL = 0
0012       95  DO 100 I = 2,NCYNDS
C
C                  ELIMINATE ARM NODES FROM CONSIDERATION
C
0013           DO 110 L = 1,M
0014           IF(IN(L,1) .EQ. I) GO TO 100
0015       110 CONTINUE
0016           KOUNT = 0
C
C                  DETERMINE IF ANY NODE MUST BE IN A CERTAIN DISTRICT
C
0017           DO 120 K = 1,M
0018           KOUNT = KOUNT + INTREE(I,K)
0019           IF(KOUNT .GT. 1) GO TO 100
0020       120 CONTINUE
C
C                  PUT NECESSARY PATH NODES IN TREE IF A NODE CAN
C                  BE IN THAT TREE ONLY
C
0021           DO 130 K  = 1,M
0022           IF(INTREE(I,K) .EQ. 1) GO TO 140
0023       130 CONTINUE
```

```
0024          140 NSOURC   = I
                C
                C              DETERMINE NTERM
                C
0025                NOK = NO(K)
0026                DO 145 JI = 1,NOK
0027                IF(DIST(I,IN(K,JI)) + DLOAD(K) .LE. C ) GO TO 146
0028          145 CONTINUE
0029          146 NTERM = IN(K,JI)
0030                CALL DECODE
0031                IN(K,NO(K) + 1) = JPATH(1)
0032                INTREE(JPATH(1),K) = 2
0033                DLOAD(K) = DLOAD(K) + CLOAD(JPATH(1))
0034                DO 150 II = 2,INUM
0035                JX(JPATH(II-1) , JPATH(II)) = K
0036                JX(JPATH(II) , JPATH(II -1)) = K
0037                INTREE(JPATH(II),K) = 2
0038                NOK = NO(K)
                C
                C              SET COST OF EDGES LEAVING PATH = INF
                C
0039                DO 148 LK = 1,NCYNDS
0040          148 DIST(JPATH(II - 1),LK) = FINF
0041                DO 155 JJ = 1,NOK
0042                IF(JPATH(II) .EQ. IN(K,JJ)) GO TO 156
0043          155 CONTINUE
0044                IN(K,NO(K) + II) = JPATH(II)
0045          150 DLOAD(K) = DLOAD(K)  + CLOAD(JPATH(II))
0046          156 NO(K) = NO(K) + II   -1
0047                NUMBER = NUMBER + II   - 1
0048                DO 157 IJ=1,NCYNDS
0049                DO 157 J = 1,NCYNDS
0050                IF(SP(IJ,J)- 0. .LT. .00001) GO TO 157
0051                DIST(IJ,J)= FINF
0052          157 CONTINUE
0053                CALL SCREEN
0054                IF(IRETRN .EQ. 1) RETURN
0055                GO TO 95
0056          100 CONTINUE
                C
                C              END OF PATH NODES
                C
0057          160 DO 170 K = 1,M
0058                IF (LL .EQ. 0) GO TO 185
0059                DO 180 I = 1,LL
0060                IF (K .EQ. IOUT(I)) GO TO 170
0061          180 CONTINUE
                C
                C              FIND DISTRICT WITH MINIMUM LOAD
                C
0062          185 DO 190 L = 1,M
0063                IF(LL .EQ. 0) GO TO 195
0064                DO 219 I = 1,LL
0065                IF(L .EQ. IOUT(I)) GO TO 190
0066          219 CONTINUE
0067          195 IF(DLOAD(K) .GT. DLOAD(L)) GO TO 170
0068          190 CONTINUE
0069                GO TO 200
0070          170 CONTINUE
0071          200 DO 210 JJ = 1,NCYNDS
0072                J = NCYNDS + 1 -JJ
                C
                C              HAS NODE (I) BEEN PUT IN ANOTHER DISTRICT
                C
0073                DO 215 I = 1,NCYNDS
0074                IF(JX(I,NODE(J)) .NE. 0) GO TO 210
0075          215 CONTINUE
                C
                C              IS THERE A ZERO CSOT ARC
                C
0076                NOK = NO(K)
0077                DO 220 III = 1,NOK
0078                II = NOK + 1 -III
0079                IF(SP(IN(K,II),NODE(J)) -0. .LT. .00001) GO TO 230
0080          220 CONTINUE
0081                GO TO 210
0082          230 IF (INTREE(NODE(J),K).EQ.1) GO TO 240
0083          210 CONTINUE
0084                GO TO 250
                C
                C              ADD ANOTHER NODE TO DISTRICT K
                C
```

```
0024          140 NSOURC   = 1
              C
              C              DETERMINE NTERM
              C
0025                NOK = NO(K)
0026                DO 145 JI = 1,NOK
0027                IF(DIST(I,IN(K,JI)) + DLOAD(K) .LE. C ) GO TO 146
0028          145 CONTINUE
0029          146 NTERM = IN(K,JI)
0030                CALL DECODE
0031                IN(K,NO(K) + 1) = JPATH(1)
0032                INTREE(JPATH(1),K) = 2
0033                DLOAD(K) = DLOAD(K) + CLOAD(JPATH(1))
0034                DO 150 II = 2,INUM
0035                JX(JPATH(II-1) , JPATH(II)) = K
0036                JX(JPATH(II) , JPATH(II -1)) = K
0037                INTREE(JPATH(II),K) = 2
0038                NOK = NO(K)
              C
              C              SET COST OF EDGES LEAVING PATH = INF
              C
0039                DO 148 LK = 1,NCYNDS
0040          148 DIST(JPATH(II - 1),LK) = FINF
0041                DO 155 JJ = 1,NOK
0042                IF(JPATH(II) .EQ. IN(K,JJ)) GO TO 156
0043          155 CONTINUE
0044                IN(K,NO(K) + II) = JPATH(II)
0045          150 DLOAD(K) = DLOAD(K)  + CLOAD(JPATH(II))
0046          156 NO(K) = NO(K) + II   -1
0047                NUMBER = NUMBER + II   - 1
0048                DO 157 IJ=1,NCYNDS
0049                DO 157 J = 1,NCYNDS
0050                IF(SP(IJ,J)- 0. .LT. .00001) GO TO 157
0051                DIST(IJ,J)= FINF
0052          157 CONTINUE
0053                CALL SCREEN
0054                IF(IRETRN .EQ. 1) RETURN
0055                GO TO 95
0056          100 CONTINUE
              C
              C              END OF PATH NODES
              C
0057          160 DO 170 K = 1,M
0058                IF (LL .EQ. 0) GO TO 185
0059                DO 180 I = 1,LL
0060                IF (K .EQ. IOUT(I)) GO TO 170
0061          180 CONTINUE
              C
              C              FIND DISTRICT WITH MINIMUM LOAD
              C
0062          185 DO 190 L = 1,M
0063                IF(LL .EQ. 0) GO TO 195
0064                DO 219 I = 1,LL
0065                IF(L .EQ. IOUT(I)) GO TO 190
0066          219 CONTINUE
0067          195 IF(DLOAD(K) .GT. DLOAD(L)) GO TO 170
0068          190 CONTINUE
0069                GO TO 200
0070          170 CONTINUE
0071          200 DO 210 JJ = 1,NCYNDS
0072                J = NCYNDS + 1 -JJ
              C
              C              HAS NODE (I) BEEN PUT IN ANOTHER DISTRICT
              C
0073                DO 215 I = 1,NCYNDS
0074                IF(JX(I,NODE(J)) .NE. 0) GO TO 210
0075          215 CONTINUE
              C
              C              IS THERE A ZERO CSOT ARC
              C
0076                NOK = NO(K)
0077                DO 220 III = 1,NOK
0078                II = NOK + 1 -III
0079                IF(SP(IN(K,II),NODE(J)) -0. .LT. .00001) GO TO 230
0080          220 CONTINUE
0081                GO TO 210
0082          230 IF (INTREE(NODE(J),K).EQ.1) GO TO 240
0083          210 CONTINUE
0084                GO TO 250
              C
              C              ADD ANOTHER NODE TO DISTRICT K
              C
```

```
0085          240 NUMBER = NUMBER +1
0086              NO(K) = NO(K) +1
0087              IN(K,NO(K)) = NODE(J)
0088              JX(IN(K,I1),IN(K,NO(K))) = K
0089              JX(IN(K,NO(K)),IN(K,I1)) = K
0090              DLOAD(K) = DLOAD(K) + CLOAD(IN(K,NO(K)))
0091              IF(NUMBER .NE. NCYNDS) GO TO 160
0092              GOT O 240
0093          250 LL = LL + 1
0094              IF(LL .GT. M) IRETRN = 1
0095              IF(LL .GT.M) GO TO 260
                C
                C             IOUT IS A LIST OF DISTRICTS WHICH CANT BE INCREASED
                C
0096              IOUT(LL) = K
0097              IF(NUMBER .NE. NCYNDS) GO TO 160
0098          260 RETURN
0099              END



0001              SUBROUTINE LABEL
                C
                C             LABEL NEGATES ALTERNATE ELEMENTS OF THE AROUND
                C             MATRIX
                C
0002              COMMON DIST(90,90),IPATH(90,90),JPATH(90),IEND(270),
                 1JEND(270),COST(270),NODD(90),ELOAD(270),NSOURC,NTERM,INUM,
                 2 NUMNDS,NUMEGS,I INODE(1080),JJNODE(1080),AROUND(90,10),NUMINC(90),
                 3 INNODE(60,10),CYCNDS(60),SP(60,60),NRSTND(60,60),CLOAD(60),
                 4 DLOAD(20),IN(20,40),NODE(90),NO(20),JX(60,60),ITIE(100),NF(100),
                 5 SPB(60),M,N,TCOST,INF,FINF,NCYNDS,IDEPOT,AVLOAD,C,NATCH,
                 6 NEXTND(10),ITOUR(20,300),NOTOUR(20) ,STOP,IMOUT,COSTK(20)
0003              INTEGER AROUND
0004              DIMENSION LBLD(90)
                C
                C             LABEL ROW 1 OF AROUND MATRIX
                C
0005              NI = NUMINC(IEND(1))
0006             45 DO 50 IJ =  1,NI,2
0007             50 AROUND(IEND(1),IJ)= -AROUND(IEND(1),IJ)
0008             65 LBLD(1) = IEND(1)
0009              NLBLD = 1
0010              KK = 1
0011              DO 90 K = 1,NUMNDS
0012              LBLDK = LBLD(K)
                C
                C             DETERMINE IF THE NEXT ENTRY IN THE  AROUND(LBLDK,J)
                C             ROW HAS HAD ITS ROW LABELED
                C
0013              NL = NUMINC(LBLDK)
0014              DO 90 I = 1,NL
0015              DO 100 J = 1,NLBLD
0016              IF(IABS(AROUND(LBLDK,I)) .EQ. LBLD(J)) GO TO90
0017            100 CONTINUE
0018              KK = KK + 1
                C
                C             RECORD THE NEXT NODE AS HAVING ITS AROUND ROW LABELED
                C
0019              LBLD(KK) = IABS(AROUND(LBLDK,I))
0020              LBLDKK = LBLD(KK)
                C
                C             FIND THE ELEMENT IN THE NEW AROUND ROW WHICH
                C             CORRESPONDS TO THE ROW FROM WHICH IT CAME
                C
0021              NLK= NUMINC(LBLDKK)
0022              DO 110 JJJ= 1,NLK
0023              JJ  = NLK+ 1 - JJJ
0024              IF(AROUND(LBLDKK,JJ ).EQ. LBLDK) GO TO 115
0025            110 CONTINUE
                C
                C             LABEL THE ELEMENTS IN THE AROUND(I,J) ROW
                C
0026            115 IF(AROUND(LBLDK,I) .GT. 0) GO TO 120
```

```
0027              NLK = NUMINC(LBLDKK)
0028              JJ1 = JJ + 1
0029              DO 125 IJ = JJ1,NLK,2
0030            125 AROUND(LBLDKK,IJ) = -AROUND(LBLDKK,IJ)
0031              IF(JJ - 1 .EQ.0) GO TO 130
0032              JJ1 = JJ - 1
0033              DO 127 IJ = 1,JJ1,2
0034              IJJ = JJ1 + 1 - IJ
0035            127 AROUND(LBLDKK,IJJ) = -AROUND(LBLDKK,IJJ)
0036              GO TO 130
0037            120 NLK = NUMINC(LBLDKK)
0038              DO 128 IJ = JJ,NLK,2
0039            128 AROUND(LBLDKK,IJ) = -AROUND(LBLDKK,IJ)
0040              IF(JJ - 2 .LE. 0) GO TO 130
0041              JJ2 = JJ - 2
0042              DO 129 IJ = 1,JJ2,2
0043              IJJ = JJ2 + 1 - IJ
0044            129 AROUND(LBLDKK,IJJ) = -AROUND(LBLDKK,IJJ)
                C
                C               HAVE ALL ROWS BEEN LABELED
                C
0045            130 NLBLD = NLBLD + 1
0046              IF (NLBLD .EQ. NUMNDS) GO TO 140
0047             90 CONTINUE
0048            140 RETURN
0049              END



0001              SUBROUTINE ORDERK
                C
                C               THIS SUBROUTINE WILL
                C               REORDER K'S SUCH THAT DLOAD(K) .LE. DLOAD (K +1)
                C
0002              COMMON DIST(90,90),IPATH(90,90),JPATH(90),IEND(270),
                 1JEND(270),COST(270),NODD(90),ELOAD(270),NSOURC,NTERM,INUM,
                 2 NUMNDS,NUMEGS,IINODE(1080),JJNODE(1080),AROUND(90,10),NUMINC(90),
                 3 INNODE(60,10),CYCNDS(60),SP(60,60),NRSTND(60,60),CLOAD(60),
                 4 DLOAD(20),IN(20,40),NODE(90),NO(20),JX(60,60),ITIE(100),NF(100),
                 5 SPB(60),M,N,TCOST,INF,FINF,NCYNDS,IDEPOT,AVLOAD,C,NATCH,
                 6 NEXTND(10),ITOUR(20,300),NOTOUR(20) ,STOP,IMOUT,COSTK(20)
0003              ITEMP = 0
0004              TEMP = 0.
0005              MM = M - 1
0006              DO 90 K = 1,MM
0007              K1 = 0
0008              TEMP = DLOAD(K)
0009              MK = M - K
0010              DO 10 J = 1,MK
0011              IF(TEMP .LE. DLOAD(K + J)) GO TO 10
0012              K1 = K + J
0013              TEMP = DLOAD(K1)
0014             10 CONTINUE
0015              IF (K1 .EQ. 0) GO TO 90
0016              DLOAD(K1) = DLOAD(K)
0017              DLOAD(K) = TEMP
                C
                C               REORDER NO(K)
                C
0018              ITEMP = NO(K1)
0019              NO(K1) = NO(K)
0020              NO(K) = ITEMP
                C
                C               REORDER IN(K,J)
                C
0021              IF(NO(K) .GE. NO(K1)) GO TO 20
0022              N2 = NO(K1)
0023              GO TO 30
0024             20 N2 = NO(K)
0025             30 DO 40 J= 1,N2
0026              ITEMP = IN(K1,J)
0027              IN(K1,J) = IN(K,J)
0028             40 IN(K,J) = ITEMP
                C
                C               REORDER JX(I,J)
                C
0029              DO 70 I= 1,N
0030              DO 70 II = 1,N
0031              IF(JX(I,II) .EQ. K) GO TO 80
0032              IF(JX(I,II).NE. K1) GO TO 70
0033               JX(I,II) = K
0034              GO TO 70
```

```
0035            80 JX(I,II) = K1
0036            70 CONTINUE
0037            90 CONTINUE
0038               RETURN
0039               END
```

```
0001               SUBROUTINE OUT
                C
                C          THIS SUBROUTINE PRINTS THE EDGES IN EACH DISTRICT, THE
                C          COST FOR EACH DISTRICT AND THE TOTAL COST
                C
                C
0002               COMMON DIST(90,90),IPATH(90,90),JPATH(90),IEND(270),
                  1JEND(270),COST(270),NODD(90),ELOAD(270),NSOURC,NTERM,INUM,
                  2 NUMNDS,NUMEGS,IINODE(1080),JJNODE(1080),AROUND(90,10),NUMINC(90),
                  3 INNODE(60,10),CYCNDS(60),SP(60,60),NRSTND(60,60),CLOAD(60),
                  4 DLOAD(20),IN(20,40),NODE(90),ND(20),JX(60,60),ITIE(100),NF(100),
                  5 SPB(60),M,N,TCOST,INF,FINF,NCYNDS,IDEPOT,AVLOAD,C,NATCH,
                  6 NEXTND(10),ITOUR(20,300),NOTOUR(20) ,STOP,IMOUT,COSTK(20)
0003               TCOST = 0.
0004               WRITE(6,995) N,M
0005            995 FORMAT('0''THIS NETWORK HAS',I3,'CYCLE NODES AND',I3,'POSTMEN')
0006               DO 100 K = 1,M
0007               WRITE(6,1000) K,ND(K),DLOAD(K)
0008           1000 FORMAT (10X'DISTRICT',I3,'HAS',I3,'CYCLE NODES WITH A TOTAL LOAD
                  COF',F10.2,'. THE EDGES INCLUDED ARE')
0009               WRITE(6,1005)
0010           1005 FORMAT ('0',11X'EDGE',12X,'COST')
0011               COSTT  =0.
0012               DO 110 I = 1,N
0013               DO 110 J = 1,N
0014               IF (JX(I,J) .NE. K) GO TO 110
0015               WRITE(6,1010) I,J,SP(I,J)
0016           1010 FORMAT(    10X'(',I3,',',I3,')',4X,F10.2)
0017               COSTT  = COSTT + SP(I,J)
0018            110 CONTINUE
0019               WRITE(6,1020) K,COSTT
0020           1020 FORMAT('0',10X'THE TOTAL NUMBER OF RETRACED  UNITS IN DISTRICT',
                  CI3,'IS',F10.2)
0021            100 TCOST  = TCOST + COSTT
0022               WRITE (6,1030) TCOST
0023           1030 FORMAT('0',10X'THE TOTAL NUMBER OF RETRACED UNITS IS',F10.2)
0024               RETURN
0025               END
```

INPUT

NUMBER OF NODES = 21
NUMBER OF EDGES = 37
NUMBER OF DISTRICTS = 4
DEPOT IS NODE 1
TRUCK CAPACITY = 78.000
TERMINATION COST = 1000.000
EPSILON = 0.0

NETWORK EDGES

EDGE	IEND	JEND	COST	LOAD
1	1	11	10.500	7.300
2	11	12	5.900	5.000
3	12	13	7.100	7.500

4	1	2	8.200	7.500
5	11	9	8.200	9.000
6	12	14	11.500	9.100
7	13	14	5.700	3.200
8	2	9	10.500	5.200
9	9	10	2.500	10.000
10	10	14	6.800	8.700
11	2	4	8.900	8.700
12	2	3	7.800	7.500
13	9	8	7.800	6.500
14	10	16	5.100	5.000
15	10	8	6.200	6.500
16	10	15	7.800	7.800
17	14	15	7.500	7.500
18	4	3	2.400	2.000
19	3	8	10.800	10.000
20	8	16	5.100	5.800
21	16	15	4.200	4.700
22	4	5	8.700	8.500
23	3	5	10.000	10.000
24	3	6	8.700	8.500
25	8	7	8.700	7.800
26	16	17	9.000	8.100
27	15	17	8.200	8.000
28	5	6	5.100	9.100
29	6	7	7.900	6.200
30	7	17	12.000	6.200
31	5	18	3.200	6.900
32	7	19	3.200	3.000
33	17	20	3.200	3.000
34	18	19	15.100	10.200
35	19	20	14.900	8.700
36	7	21	2.000	2.000
37	21	19	1.900	1.900

AROUND MATRIX

NODE	INCIDENT NODES				
1	11	2			
2	1	4	3	9	
3	2	4	5	6	8
4	2	5	3		
5	4	18	6	3	
6	3	5	7		
7	6	19	21	17	8
8	3	7	16	10	9
9	2	8	10	11	
10	9	8	16	15	14
11	1	9	12		
12	11	14	13		
13	12	14			
14	12	10	15	13	
15	10	16	17	14	
16	10	8	17	15	
17	15	16	7	20	
18	5	19			
19	21	7	18	20	
20	17	19			
21	7	19			

```
TOUR  1

   TRAVEL FROM NODE  1
     TO NODE(S)  2  3
   SERVICE FROM NODE  3
     TO NODE(S)  8  7 17 20 19 21  7  6  3
   TRAVEL FROM NODE  3
     TO NODE(S)  2  1

   TRAVELED DISTANCE =      32.00
   LOAD =      54.30


TOUR  2

   SERVICE FROM NODE  1
     TO NODE(S) 11  9 10 16  8
   TRAVEL FROM NODE  8
     TO NODE(S) 10
   SERVICE FROM NODE 10
     TO NODE(S)  8  9  2  1

   TRAVELED DISTANCE =       6.20
   LOAD =      60.80


TOUR  3

   TRAVEL FROM NODE  1
     TO NODE(S)  2
   SERVICE FROM NODE  2
     TO NODE(S)  3  5  6
   TRAVEL FROM NODE  6
     TO NODE(S)  7
   SERVICE FROM NODE  7
     TO NODE(S) 19 18  5  4
   TRAVEL FROM NODE  4
     TO NODE(S)  3
   SERVICE FROM NODE  3
     TO NODE(S)  4  2
   TRAVEL FROM NODE  2
     TO NODE(S)  1

   TRAVELED DISTANCE =      26.70
   LOAD =      65.90


TOUR  4

   TRAVEL FROM NODE  1
     TO NODE(S) 11
   SERVICE FROM NODE 11
     TO NODE(S) 12 13 14 15 17 16 15 10 14 12
   TRAVEL FROM NODE 12
     TO NODE(S) 11  1
   TRAVELED DISTANCE =      26.90
   LOAD =      69.60
```

B. NOTES ON THE COMPUTER CODE

Listed below are the major variables used throughout the program with an explanation for each one. Variables used exclusively in the KMATCH and OUTPUT subroutines are not included in the list.

IEND(I) - the front end of edge i

JEND(I) - the tail end of edge i

COST(I) - the length of edge i

ELOAD(I) - the load on edge i

NUMNDS - the number of nodes in the original network

NUMEGS - the number of edges in the original network

M - the number of desired districts

C - truck capacity

IDEPOT - the node representing the depot

AROUND(I,J) - the Jth node adjacent to node i

INF, FINF - infinity

NODE(I) - nodes listed in order of distance from the depot

IMOUT - a flag indicating whether intermediate output is desired (1 - yes, 0 - no)

DIST(I,J) - shortest path lengths between i and j on the original and unicursal networks

STORE(I,J) - stored edge lengths between i and j before the shortest path routine is run

AVLOAD	- the average load per truck
STOP	- objective function value above which the algorithm will terminate
NODD(I)	- list of odd nodes
NUMINC(I)	- the number of edges incident with node i
JPATH(I)	- sequence of nodes in a specific shortest path
IPATH(I,J)	- next node in the shortest path sequence from i to j
NSOURC	- the front end of a path
NTERM	- the tail end of a path
SP(I,J)	- the shortest path between cycle node i and cycle node j.
IINODE(I), IKM(I)	- front end of edge i input to KMATCH
JJNODE(I), JKM(I)	- tail end of edge input to KMATCH
MATCH(I)	- flag to indicate if edge i is matched
EDGWT(I)	- cost of edge i input to KMATCH
NCYNDS	- the number of cycle nodes
INNODE(I,J)	- the jth node in cycle node i
CYCNDS(I)	- the number of nodes in cycle node i
NRSTND(I,J)	- the node in cycle node j that nearest cycle node i
CLOAD(I)	- the load on cycle node i
DLOAD(I)	- the load on district i

IN(I,J)	- the jth cycle node in district i
NO(I)	- the number of cycle nodes in district i
JX(I,J)	- a flag indicating whether edge (i,j) is in the capacitated spanning tree
INTREE(I,J)	- a flag indicating if cycle node i can be reached from the arm for district j.
ALIST(I)	- costs of possible combinations of arms
IPACK(I,J)	- packed word indicating what arms are being considered for combination i
MASK(I)	- packed word for interpreting IPACK(I,J)
NEXTND(I)	- a list of possible cycle nodes to branch to when decoding
NEXT(I)	- the next element in ALIST(I,J) after element i
ITOUR(I,J)	- the jth node in tour i
NOTOUR(I)	- the number of nodes in tour i

In order to change the dimensions of the program the dimensioned variables are categorized below:

Dimensioned at the number of nodes are: DIST, IPATH, JPATH, NODD, NODE, STORE, LBLD

Dimensioned at the number of cycle nodes, (approximately 2/3 of the number of nodes): CYCNDS, SP, NRSTND, CLOAD, JX, SPD, INJX

Dimensioned at the number of edges, (3 times the number of nodes): IEND, JEND, COST, ELOAD

Dimensioned at the number of districts: DLOAD, NOTOUR, COSTK

Dimensioned at K, where K = the number of odd nodes: IINODE, JJNODE, MATCH, IKM, JKM, EDGWT

Dimensioned at the number of nodes and 10: AROUND

Dimensioned at the number of cycle nodes and 10: INNODE

Dimensioned at the number of districts and an independent number: IN, ITOUR, INTREE

Dimensioned at an independent number: ITIE, NF, NEXTND, ALIST, NEXT, IPACK, MASK

If the dimensioned size is to be increased to where more than 60 cycle nodes are anticipated then IPACK(I,J) must have J increased proportionately. Slight modifications to the code will also be necessary.

Three function subroutines that are used in the ARMS subroutine are not standard library subroutines. IORZ, IANDZ, and IXORZ are University of Illinois routines and are not standard with all computers.

The KMATCH and OUTPUT subroutines were coded by White (34) and are a modification of an algorithm developed by Edmonds (9).

C. INPUT AND OUTPUT

Listed below are the various parameters which must

be input to the algorithm. Specific format details are shown at the beginning of section one of the listing.

1. the number of nodes in the original network
2. the number of edges in the original network
3. the number of districts desired
4. the node which represents the depot
5. truck capacity
6. a termination cost. If the value at the objective function exceeds this cost the algorithm will terminate
7. intermediate output indicator. Intermediate output will be printed if a one is input
8. a value which increases the value of capacity. Solutions will be printed which are feasible for this larger value of capacity. The algorithm will, however, proceed to try and find a solution feasible to the original value of capacity.
9. criterion 2 indicator. Criterion one will be used solely to exchange cycle nodes in sections unless a one is input to indicate that the use of criterion two is also desired.
10. for each edge, the front end node, the tail end node, the edge cost and the edge load. Edge number one must be on the extremity of the

network and input such that when traveling from the front end to the tail end the network is to the left. This edge must also be the only edge joining IEND(I) and JEND(I).

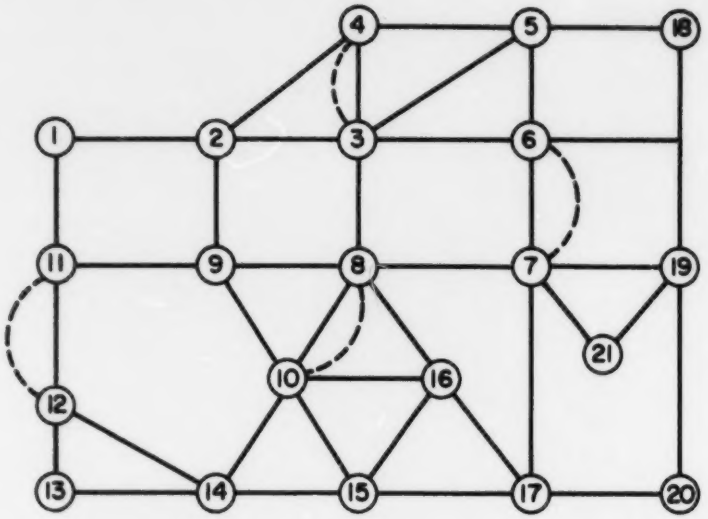
11. for each node a list of the adjacent nodes. The node must be listed in clockwise order. The first node adjacent to IEND(I) must be JEND(I), i.e. AROUND(IEND(I) , I) = JEND(I).

The algorithm automatically lists the input. Intermediate output will be listed if the user desires it, and includes matrices at various points during the algorithm. The information listed includes the edges after matching (IEND, JEND, COST, LOAD), AROUND(I,J), NUMINC(I), SP(I,J), INNODE(I,J), CYCNDS(I), NRSTND(I,J), CLOAD(J), and a list of attempted arms.

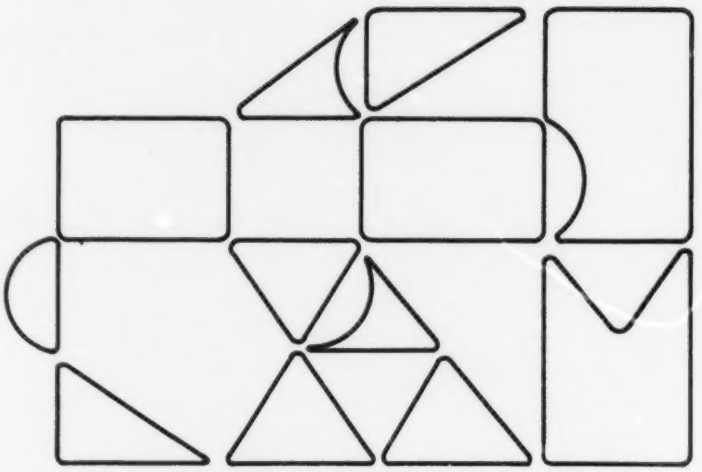
Final output is a list of nodes included in each tour. The algorithm indicates whether the truck must travel or service from one node to another.

D. EXAMPLE

Figure 8 illustrates the various steps for an example problem. The input and output for the problem are listed at the end of the program listing in section A of the appendix.



(a) Original Network (Solid Lines) and Duplicated Edges (Dashed Lines) to Create Unicursal Graph



(b) Cycles Designated on Unicursal Graph

FIGURE 8. STEPS IN SOLVING THE EXAMPLE PROBLEM

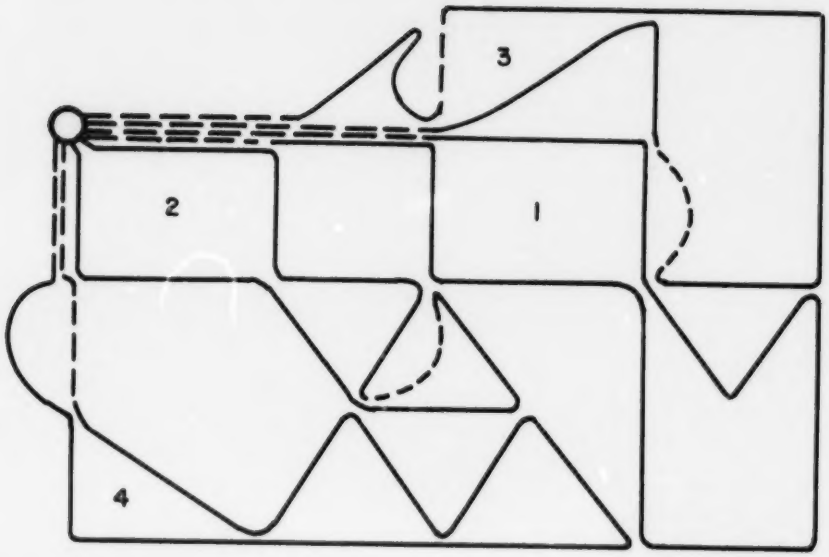


FIGURE 8c. FINAL TOURS OF EXAMPLE PROBLEM

REFERENCES

1. Bellman, Richard and K. L. Cooke, "The Königsberg Bridges Problem Generalized," Jour. of Math. Anal. and Appl. 25, pp. 1-7, 1969.

2. Beltrami, Edward J., and Lawrence D. Bodin, "Networks and Vehicle Routing for Municipal Waste Collection," State University of New York at Stony Brook, Urban and Policy Sciences, Report No. UPS 72-18, August, 1972.

3. Berge, Claude, The Theory of Graphs, John Wiley and Sons, Inc., New York, 1962.

4. Black, R. J., "The Solid Waste Problem in Metropolitan Areas," California Vector Views, 11, September, 1964.

5. Bodner, R. M., E. A. Cassell, and P. J. Andros, "Optimal Routing of Refuse Collection Vehicles," Jour. of the San. Eng. Div., ASCE, 96, SA4, pp. 893-904, August, 1970.

6. Brown, Robert S., "An Analytic Model for District Assignment in Solid Waste Collection," M. S. Thesis, Department of Civil Engineering, Mass. Inst. of Tech., Cambridge, Mass., February, 1972.

7. Clark, G., and J. W. Wright, "Scheduling of Vehicles from a Central Depot to a Number of Delivery Points," Operations Research, 12, pp. 568-581, 1964.

8. Dreyfus, S. E., "An Appraisal of Some Shortest Path Algorithms," RM 5433-PR, The Rand Corporation, Santa Monica, Calif., October, 1967.

9. Edmonds, Jack, "The Chinese Postman's Problem," Bulletin of ORSA, vol. 13, Supplement 1, p. B-73, 1965.

10. Edmonds, Jack, "Maximum Matching and a Polyhedron with 0,1-Vertices," Jour. of Research, National Bureau of Standards, vol. 69B, pp. 125-130, January-June, 1965.

11. Edmonds, Jack, "Paths, Trees, and Flowers," Canad. Jour. of Math., vol. 17, pp. 449-467, 1965.

12. Edmonds, Jack, "An Introduction to Matching," Mimeographed Lectures, University of Michigan Summer Eng. Conf., Ann Arbor, 1967.

13. Edmonds, Jack and Ellis L. Johnson, "Matching: A Well-Solved Class of Integer Linear Programs," in Combinatorial Structures and Their Applications, Gordon and Breach, pp. 89-92, 1970.

14. Edmonds, Jack, unpublished, untitled report, National Bureau of Standards, March, 1970.

15. Edmonds, Jack and Ellis L. Johnson, "Matching, Euler Tours, and the Chinese Postman," to appear, Mathematical Programming.

16. Euler, Leonhard, "Solutio Problematis ad Geometriam Situs Pertinentis," Comment. Academiae Sci. I. Petropolitanae. 8, 1736, pp. 66-70.

17. Ford, L. R., Jr. and D. R. Fulkerson, Flows in Networks, Princeton University Press, 1962.

18. Garfinkel, Robert S. and George L. Nemhauser, Integer Programming, John Wiley and Sons, N. Y., 1972.

19. Hadley, G., Linear Programming, Addison-Wesley Publishing Company, Inc., 1962.

20. Hong, Saman, "A Linear Programming Approach for the Traveling Salesman Problem," Ph.D. Dissertation, The Johns Hopkins University, 1972.

21. Johnson, Ellis L., "Existence of Euler Tours in Bidirected Graphs," IBM Research, RC 2753, January, 1970.

22. Kwan Mei-Ko, "Graphic Programming Using Odd or Even Points," Chinese Math., 1, pp. 273-277, 1962.

23. Liebling, T. M., "Graphentheorie in Planungsund Tourenproblemen," Lecture Notes in Operations Research and Mathematical Systems, 21, Springer-Verlag, 1970.

24. Liebman, Jon C., "Notes on Edmond's Matching and White's K-Matching," unpublished class notes, June, 1972.

25. Liebman, Jon C., "Routing of Solid Waste Collection Vehicles," Final Report of EPA Research Grant 801289, University of Illinois at Urbana-Champaign, August, 1973.

26. Marks, D. H., "Facility Location and Routing Models in Solid Waste Collection Systems," Ph.D. Dissertation, The Johns Hopkins University, 1969.

27. Marks, D. H. and J. C. Liebman, Discussion of "Optimal Routing of Refuse Collection Vehicles," by R. M. Bodner, E. A. Cassell and P. J. Andros, Jour. of the San. Eng. Div., ASCE, SA1, pp. 130-131, February, 1971.

28. Orloff, Clifford, "Routing a Fleet of M Vehicles to/from a Central Facility," Cornell University, Ithaca, N. Y., November, 1972.

29. Owen, E. H., "Computer Program Cuts Cost of Urban Solid Waste Collection," Public Works, January, 1970.

30. Shuster, Kenneth A., and Dennis A. Shur, "A Heuristic Approach to Routing Solid Waste Collection Vehicles," U. S. Environmental Protection Agency, Office of Solid Waste Management Programs, September, 1972.

31. Stricker, Robert, "Public Sector Vehicle Routing: The Chinese Postman Problem," M. S. Thesis, Dept. of Electrical Engineering, Mass. Inst. of Tech., Cambridge, Mass., August, 1970.

32. Truitt, M. M., J. C. Liebman, and C. W. Kruse, "An Investigation of Solid Waste Collection Policies," vol. 1, Dept. of Environmental Eng. Science, The Johns Hopkins University, Baltimore, Maryland, August, 1968.

33. Wathne, Magne, "Optimal Routing of Solid Waste Collection Vehicles," Ph.D. Dissertation, The Johns Hopkins University, 1972.

34. White, Lee J., "A Parametric Study of Matchings and Coverings in Weighted Graphs," Doctoral Thesis, Tech. Rep. 06920-11-T, Systems Engineering Laboratory, Dept. of Electrical Engineering, University of Michigan, Ann Arbor, 1967.